Design and Implementation of IoT Android Commissioner

Andy Lianghua Xu, Jan Janak, Henning Schulzrinne

Abstract

As Internet of Things (IoT) devices gain more popularity, device management gradually becomes a major issue to IoT device users. To manage an IoT device, the user first needs to join it to an existing network. Then, the IoT device has to be authenticated by the user. The authentication process often requires a two-way communication between the new device and a trusted entity, which is typically a handheld device owned by the user. To ease and standardize this process, we present the Device Enrollment Protocol (DEP) as a solution to the enrollment problem described above. Starting from DEP, we then showcase the design of an IoT device commissioner and its prototype implementation on Android, named *Android Commissioner*. The application allows the user to authenticate IoT devices and join them to an existing protected network.

1 Introduction

The emergence of Internet of Things (IoT) devices gives rise to many device management issues. For instance, smart-home owners often need to manage different types of IoT devices separately, such as smart locks, smart light bulbs, thermostats, smart gas heaters. Each IoT device typically has its own setup procedure and custom management tools provided by the manufacturer. As a result, users typically need to spend a significant amount of time learning how to configure such device. For example, to set up a newly bought IoT device, the device owner first needs to physically install the device, join the device to an existing Wireless Local Area Network (Wi-Fi), and use the application provided by the manufacturer to configure the device. In most cases, the manufacturer only provides a custom app to manage one particular type of IoT device. Therefore, the end user often needs to download and install an app for each IoT device type in order to enroll a heterogeneous mix of IoT devices.

The manufacturer-specific way of enrolling IoT devices poses several problems. First, proprietary enrollment protocols without wider community review often contain security vulnerabilities. Second, the need to keep multiple copies of credentials across many third-party enrollment apps increases the risk that the credentials are stolen. Third, the enrollment process of IoT devices often requires user's physical interactions. This is because for small IoT devices with no user interface, it's often hard to configure or change Wi-Fi passwords, hence a physical button is used to serve the same purpose of authenticating the IoT device. However, physical interaction presents a challenge for many types of devices that are out-of-reach, e.g., devices that are part of building infrastructure. Last problem but not the least, transferring IoT device ownership is also a difficult task for users that purchase second-hand IoT devices. Such user typically needs to repeat the entire enrollment procedure without the possibility to reuse some of the (tedious and time-consuming) steps performed by the previous user.

We propose an enrollment system for IoT devices that attempts to address these issues. Our system is capable of enrolling multiple IoT devices into an existing network at the same time. We also propose the Device Enrollment Procedure (DEP), a protocol we design to be used in the enrollment system. DEP is a general, easy-to-implement protocol that facilitates communication inside the enrollment system. The enrollment system involves two entities: one is called the *commissioner*, which confers network credentials to the IoT devices and enrolls them to an existing network. This report mainly discusses an Android implementation of the commissioner, named the *Android Commissioner*. The other entity of the enrollment system is called the *Enrollment Daemon*, an application that is pre-installed on IoT devices that receive the credentials from the commissioner. The commissioner and the Enrollment Daemon use DEP to communicate with each other. The goal of the enrollment system is to makes it possible to securely authenticate and manage multiple IoT devices.

This report provides an overview of DEP and the Android Commissioner app. In Section **??**, we provide a brief overview of related protocols and technologies. Section **??** reviews related work. Section **??** discusses the overall design of our enrollment system. In Section **??**, we present the details of our prototype implementation. Finally, we conclude in Section **??** and discuss future plans and improvements.

2 Background

2.1 Wi-Fi Direct

Wi-Fi Direct, certified by Wi-Fi Alliance, is a protocol that enables Wi-Fi devices to establish network connections directly [**?**, **?**]. Wi-Fi Direct provides features that are very useful in the context of IoT devices. It allows the creation of a basic service set (BSS) under IEEE 802.11 without the need to install a dedicated Wi-Fi access point. Any Wi-Fi Direct enabled device can function as an access point (AP), whereas any Wi-Fi capable device can function as a station (STA).

Wi-Fi Direct allows multiple devices to form an IEEE 802.11 group, or a Wi-Fi P2P group. Depending on its role in the group, a device can either be an AP or an STA. The AP device is the creator and manager of the group, whereas other devices are all STAs. All devices in the same Wi-Fi P2P group fall under the same Wi-Fi, in which they must communicate with each other via the AP. To uniquely identify a Wi-Fi P2P group, its SSID is configured automatically when the group is created. According to the specification published by the Wi-Fi Alliance, a Wi-Fi P2P group is named in the following format: *DIRECT-ID* [**?**], where *ID* is a unique identifier of the group.

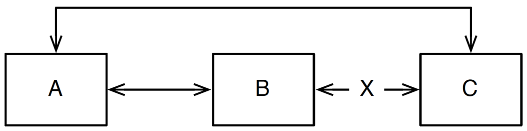


Figure 1: The Hidden Node Problem. Three devices (A, B and C) try to form a Wi-Fi P2P group by negotiation. Suppose A and B negotiate and B is chosen as the AP device of the group. If A tries to invite C to the group but C cannot communicate with B due to some physical obstacles, then the invitation will fail and C will not join the group.

Wi-Fi Direct requires a group to be formed via negotiation, in which two or more devices negotiate their roles in the group. Each device announces its intent, an integer ranging from 0 to 15. The intent number is randomly chosen by default, but can be explicitly overwritten by the application. The device with greater intent value becomes the AP of the group. Any group member (AP or STAs) can invite other devices to join the group as STAs. This negotiation scheme has benefits and weaknesses. The upside is that it always selects the device that is capable of becoming the AP without user intervention. The downside of the negotiation scheme is commonly known as the hidden node problem [**?**]. Consider the following case shown in Figure **??**.

There are three wireless devices - A, B and C. Assume that there's an obstacle between B and C so that B and C cannot communicate directly. Suppose A and B negotiate to form a Wi-Fi P2P group. Eventually, B becomes the AP of the group. If A tries to invite C to the group, the process will fail because C is unable to communicate with B, the AP of the group. This example shows that the negotiation scheme does not guarantee to work if the group consists of more than two devices. On the other hand, the negotiation scheme works well if the goal is to form a simple adhoc network of two devices in an RF environment without obstacles. For example, it is used widely for, e.g., connecting a laptop to a printer temporarily, or connecting a smartphone to a digital picture frame to transfer pictures.

Although Wi-Fi Direct supports more than 2 devices, it has been primarily designed to serve peer-to-peer communication between 2 devices, such as simple data transfer from a sender to a receiver. A Wi-Fi P2P group with more than 2 devices suffers from the hidden node problem. However, we can avoid the hidden node problem if we have an AP device that can be moved around freely. In the context of this report, the movable device is the so-called commissioner. Thus, DEP requires the commissioner to always become the AP of the group after the negotiation process. Once a Wi-Fi P2P group is formed, other devices can be invited by the commissioner to join as STAs.

In late 2016, Wi-Fi Direct is supported by all Android devices running Android 4.0 or above [?], which covers more than 98% of the Android devices on the market [?]. It is also compatible with Apple's Multipeer Connectivity Module in iOS [?].

2.2 Zeroconf

Zero-Configuration networking (Zeroconf) is a set of protocols that can be used to create automatically configured TCP/IP networks [?, ?]. The most prominent feature of Zeroconf is that it does not require a network administrator to configure the network manually, nor any special device to setup the network.

The three major goals of Zeroconf are the assignment of network addresses, the distribution and resolution of network hostnames, and the automatic discovery of network services. With Zeroconf, the network can be set up automatically without using the Dynamic Host Configuration Protocol (DHCP) or relying on a Domain Name System (DNS) server [?, ?].

DNS Service Discovery (DNS-SD) provides the service discovery feature that Zeroconf requires by using a multicast-DNS protocol [?, ?]. When a Zeroconf enabled device joins a network, it optionally publishes services it supports by registering the following attributes in the local multicast DNS responder:

- The hostname and port number of the service it provides.
- Service name, e.g., HP Printer
- Service type, e.g., _printer._tcp
- Other optional attributes, such as a unique serial number pre-assigned by the device manufacturer.

3 Related Work

There are many applications in the market that support device authentication and credential transfer. For example, Google has its own Android and iOS application to authenticate Google Chromecast and transfer network credentials over Wi-Fi [?]. Amazon also has its own applications that serve similar purposes for Amazon Echo [?]. Although these credential commissioning applications are robust and secure, they are not reliable and interoperable. For example, the application to enroll Google Chromecast cannot be used to enroll Amazon Echo.

There have been efforts to authenticate and join Wi-Fi devices under one bootstrap process, such as Wi-Fi Protected Setup (WPS) by Wi-Fi Alliance [?] and AllJoyn on-boarding process [?].

WPS allows wireless devices to be authenticated and conferred wireless credentials with only one button push. Nevertheless, several security vulnerabilities had been recently discovered and this eventually led to the deprecation of WPS [?]. Also, WPS is a single-domain solution designed for Wi-Fi and does not work with other link layer technologies such as Bluetooth, NFC, etc.

AllJoyn on-boarding process, proposed by the AllSeen Alliance, is another framework that provides a user-friendly way of enrolling new devices onto an existing network [?]. However, it also has several drawbacks and unaddressed issues. First, it is an integrated solution rather than a standalone one. The enrollment process requires both parties (the enroller and the enrollee) to have the AllJoyn framework ready and related services deployed. Second, the AllJoyn on-boarding frameworks limits the enrollment to one device at a time. Third, the current standard of AllJoyn on-boarding framework only works on Wi-Fi networks and is thus not extensible as well.

Our goal is to design and implement a standalone enrollment framework that enables secure, fast and reliable enrollment of multiple devices to an existing network. To the best of our knowledge, there are no existing solutions that serve to safely and conveniently authenticate various kinds of IoT devices across different technology domains.

4 Application Architecture

The commissioning process has four steps: connect to IoT devices, discover services, authenticate devices and transfer credentials. DEP specifies the protocol used in each step. Correspondingly, Android Commissioner, an implementation of the commissioner on Android OS, also has four modules: Wi-Fi Direct module, Zeroconf service discovery module, public-key authentication module and credential transfer module. Each module is described in detail in the following sections.

4.1 Wi-Fi Direct Module

IoT device discovery and connection is implemented using Wi-Fi Direct, as introduced in Section **??**. Wi-Fi Direct provides a standardized solution for the Android Commissioner to discover nearby IoT devices and establish an auto-configured network with those devices.

In order to communicate with other IoT devices, the Android Commissioner creates a Wi-Fi P2P group. We always want the commissioner to be the group owner, as explained in the previous section. The only way that guarantees the commissioner to become the group owner is to create a Wi-Fi P2P group by itself. First, the Android Commissioner sets its Group Owner Intent to 15 and then creates a Wi-Fi P2P group without any other devices. Then, as the group owner (and also the only device in the group), the commissioner can invite other IoT devices to join the group as STAs.

We perform the following steps to connect to IoT devices using Wi-Fi Direct.

Part I: Standard 802.11 steps:

1. IoT devices listen to broadcast beacons from the Android Commissioner.
2. The Android Commissioner is set to discovery mode, broadcasting beacon messages.
3. IoT devices detect beacons from the commissioner and respond with a message containing device information.
4. The Android Commissioner shows a list of discovered IoT devices to the user.

Part II: DEP-specific steps:

1. The user picks a set of devices from the list to connect to.
2. The Android Commissioner creates a Wi-Fi P2P group and invites selected IoT devices to join the group.
3. On each user-selected IoT device, the Enrollment Daemon accepts the invitation from Android Commissioner and gets ready to join the group.
4. The commissioner asks the user to confirm the joining of selected IoT devices to the group.
5. A Wi-Fi P2P group is formed successfully.

Notice that Wi-Fi devices negotiated via Wi-Fi Direct are supposed to authenticate each other via WPS before they establish a network connection, which requires either a physical button press or a passphrase entered by the user, as explained in Section **??**. We disable WPS because our IoT devices often do not have a physical button that solely serves for authentication purposes, nor do they have UI capabilities to implement WPS authentication via passphrase. Disabling WPS does not compromise security in our use case since we do not rely on WPS to secure the network layer and above. In fact, the transport layer is secured by TLS, which we shall discuss in Section **??**. Thus we do not need to rely on any link layer security mechanism at all.

As a part of WPS requirement, Android OS must ask for user's authorization when another devices joins the Wi-Fi P2P group managed by this Android device. Because we don't need to rely on WPS to enforce link layer security, the user should mindlessly confirm and allow the device to join regardless. This will not compromise security because TLS can instead do the same job of authenticating the device.

When the commissioner completes the commissioning process, it can simply disconnect from all devices. The temporary Wi-Fi P2P group is destroyed when the commissioner stops functioning as the AP.

We discuss the implementation details of Wi-Fi Direct on Android Commissioner in Section **??**.

4.2 Zeroconf Service Discovery Module

Every time an IoT device joins the Wi-Fi P2P group, it configures itself with an automatically generated IP address (public, private, or link-local). For this reason, the commissioner does not know in advance what the device's IP address is. Only when given a device's IP address can the commissioner communicate with the device via TCP/IP.

Our solution is to use DNS-SD, a service discovery protocol that we previously discussed in Section **??**. Using DNS-SD, the commissioner can discover a set of IoT devices that can participate in the commissioning service. This discovery step yields a set of hostnames, which are then translated into IP addresses using multicast DNS. The exact sequence of steps is as follows:

1. The commissioner discovers IoT devices offering *_enrolled._udp*, our commissioning service, by querying the DNS for a PTR record with that service name.

2. DNS returns zero or more SRV records of the form *_enrolled._udp.[domain]*, in which *domain* is some unique hostname auto-configured by each IoT device independently.

3. The commissioner then resolves the *domain* field it receives to a set of IP addresses by querying the multicast DNS for an IPv6 record.

4. The multicast DNS responder on each IoT device returns the device's IP addresses.

5. The commissioner can connect to each IoT device via one of the IP addresses obtained from such device.

When an IoT device joins the group, the Enrollment Daemon on the device publishes information about its services in DNS SRV records [**?**]. A sample DNS SRV record is shown below.

- Record: *_enrolled._udp.example.com.*
- Port number: 1201
- TTL: 86400
- Type: SRV
- Target: *doorlock.example.com.*

The commissioner starts probing commissioning service providers as soon as a Wi-Fi P2P group is formed. When the commissioner discovers a new device providing the commissioning service, the commissioner notifies the user that it is ready to authenticate that device. Once the authentication process is initiated by the user, the commissioner sends a message to the selected device, telling it to initiate a TLS connection to the commissioner.

Details on how Zeroconf is implemented is discussed in Section **??**.

4.3 Public-key Authentication Module

When the enrollment daemon on an IoT device receives an authentication request message, it starts the authentication process by initiating a TLS session to the commissioner. The authentication process is a two-way handshake, in which the commissioner first presents its server certificates to the IoT device, which then presents its client certificates for the commissioner to verify. Client certificates are signed by authorities trusted by the commissioner and also contains IoT device specific identifiers that are preloaded on the commissioner, so that the commissioner can present them to the user who can then verify by inspection. Only after the commissioner verifies both the identity of the IoT device and the validity of the client certificates can the credential transfer process be initiated. database. See Figure **??**.

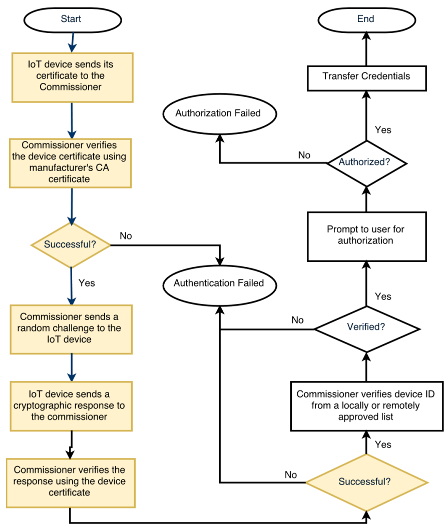

Start
IoT device sends its certificate to the Commissioner
Commissioner verifies the device certificate using manufacturer's CA certificate
Successful?
No
Yes
Commissioner sends a random challenge to the IoT device
IoT device sends a cryptographic response to the commissioner
Commissioner verifies the response using the device certificate
Successful?
No
Yes
Authentication Failed
Commissioner verifies device ID from a locally or remotely approved list
Verified?
No
Yes
Prompt to user for authorization
Authorized?
No
Yes
Authorization Failed
Transfer Credentials
End

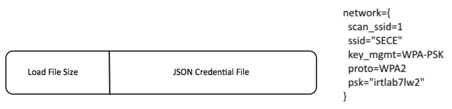
Figure 2: This figure shows the Public-Key authentication flowchart. The module runs in a finite state machine (FSM) and the flowchart can be seen as its state transition diagram. Blocks highlighted in yellow are part of standard TLS handshake procedures, whereas other are part of DEP specified procedures.

A detailed implementation of public-key authentication using Oracle's JavaX SSL/TLS library is explained Section **??**.

4.4 TLS Credential Transfer Module

We use the session generated by the TLS [**?**] handshake to securely transfer the credentials from the commissioner to the Enrollment Daemon.

The commissioner uses a simple format for the credential file based on the configuration file format of *wpa_supplicant*, a network authentication application for UNIX-like operating systems. The commissioner first sends a 4-byte integer header encoded in little-endian format indicating the size of the payload, and then followed by the payload in JSON.



```
network={
 scan_ssid=1
 ssid="SECE"
 key_mgmt=WPA-PSK
 proto=WPA2
 psk="irtlab7lw2"
}
```

Figure 3: Configuration file format used in DEP. In the left figure, the left block indicates the file size of the payload and the right block contains the JSON credential file (the payload). The right figure shows a sample JSON credential file.

5 Implementation

Below we introduce the core classes and libraries in the Android SDK that implement the required functionalities and protocols. A brief walk-through of the application framework is discussed here so that with this documentation, the application is easily extensible and maintainable by developers. Since the application is comprised of several modules, we first elaborate on the `WifiP2pManager` class, which provides application level APIs for managing Wi-Fi P2P connectivity. We then discuss the `NsdManager` class, which provides APIs for discovering services on joined networks. Lastly, we describe how the TLS server and the certificate exchange module are implemented on Android.

5.1 WifiP2pManager Class

`WifiP2pManager` is a library from Android SDK which provides the interface for applications to interact with the Wi-Fi Direct module on Android [**?**]. When the `MainActivity` of Android Commissioner is created, a handler instance of `WifiP2pManager`, named `mManager`, is obtained using `getSystemService()`. All controlling commands shall be invoked via accessing the member functions of `mManager`.

When the user toggles on the discovery mode, `mManager.discover()` is called and an instance of `WiFiBroadcastReceiver` gets created and bound to `mManager`. Once a Wi-Fi Direct device is discovered, `WiFiBroadcastReceiver` receives a callback. Information pertaining to discovered devices are stored in `WifiP2pDevice` objects.

Before connecting to the device, we have to first create a `WifiP2pGroup` by calling `mManager.createGroup()`. Then by invoking `mManager.connect(deviceInfo)`, an invitation to join the group is sent to the corresponding Wi-Fi P2P device. We can repeat this process for as many Wi-Fi P2P devices as the user hopes for. Once invitations are accepted, the group formation is completed. We then have a fully functioning `WifiP2pGroup`.

To monitor the status of a `WifiP2pGroup`, `mManager.requestGroupInfo()` can be called to obtain various information about the `WiFiP2pGroup` the commissioner has created, including a list of joined device, the group owner information and other relevant details.

`mManager.removeGroup()` is called to destroy the group and disconnect the commissioner from all devices.

5.2 NsdManager Class

Android SDK offers `NsdManager` for applications to discover DNS-based services [?]. Similar to `WifiP2pManager`, `NsdManager` is initialized using `getSystemService()`. To begin the discovery process, we create a `NsdManager.DiscoveryListener()` object and wait for its callback upon finding a new service in `WiFiP2pGroup`. Detailed information about a discovered service is wrapped into an `NsdServiceInfo` object. However, this object does not contain the address and port number of the discovered service. To obtain the address and port number, `mNsdManager.resolveService()` must be called with the `NsdServiceInfo` object passed in as parameter. The callback to `mNsdManager.resolveService()` should contain the relevant information we need, including the service's address and port number. At this point, we have a list of services with their names, addresses and port numbers.

5.3 TLS Server and Certificate Exchange

We use *Java Secure Socket Extension* (JSSE) library to implement the certificate exchange mechanism and the TLS Server [?]. The JSSE library uses a `KeyStore` object to store both the server certificate and the private key in BKS format. The library uses a `Certificate` object to store the DER encoded CA certificate in X.509 format. A `KeyStore` object and a `Certificate` object together are used to generate a `SSLContext` object using `TrustManagerFactory` and `KeyManagerFactory`.

To create a secure socket object, the JSSE library uses `SSLServerSocketFactory` with the `SSLContext` object as its parameter. A `SSLServerSocket` is generated and can be used just as a regular Java `Socket` object to accept incoming connections.

To turn on client authentication, `SSLServerSocket.setNeedClientAuth()` needs to be toggled to `true`. More information can be found in the documentation of the JSSE library [?].

6 Conclusion and Future Work

DEP, implemented on both Android Commissioner and Enrollment Daemon, provides a viable and stable solution to authenticate and join IoT devices to Wi-Fi access points (AP). Since we only use open protocols such as Wi-Fi Direct, Zeroconf, Public-Key Authentication and TLS, manufacturers can easily adapt our solution on existing hardware.

However, Android Commissioner is still a technical preview rather than a user-friendly application. For instance, the Android Commissioner presents to the user IP addresses and MAC addresses to identify IoT devices, which may be confusing to users who do not have any technical background. Moreover, when the application quits, it forgets the authentication state of all IoT devices.

To address the issues above, we plan improve the application to make it more user-friendly. On user experience side, we believe that one-click authentication makes the whole process much easier for common users. Scan and authentication process should run securely in the back-end without much user interaction.

We can also make a lot of improvements on the application. We can utilize device vendors' data such as electronic receipts (of these IoT devices) as device identifiers to simplify the authentication process. For instance, the commissioner application can load configuration files attached to online invoice to authenticate IoT devices purchased from Amazon, based on device certificate signatures provided by the vendors. We can also link the commissioner to a local or cloud database to keep track of IoT device authentication states.

We believe that these extra features are essential for Android Commissioner to be a shippable product for common users to manage IoT devices in real life.