Technical Report

# Wi-Fi Data Transmission Android & iOS Interoperability


Josef Ressel Center for
User-Friendly Secure Mobile Environments

University of Applied Sciences Upper Austria
Softwarepark 11, 4232 Hagenberg, Austria

**Author:** Martin Hengstberger hengstberger@ins.jku.at

This work has been carried out within the scope of “u’smile”, the Josef Ressel Center for User-Friendly Secure Mobile Environments, funded by the Christian Doppler Gesellschaft, A1 Telekom Austria AG, Drei-Banken-EDV GmbH, LG Nexera Business Solutions AG, NXP Semiconductors Austria GmbH, and Österreichische Staatsdruckerei GmbH. Moreover, this work has been carried out in cooperation with the Institute of Networks and Security at the Johannes Kepler University Linz.


3 Banken EDV






Revision 0.4
February 22, 2017


UNIVERSITY
OF APPLIED SCIENCES
UPPER AUSTRIA

# 1. Overview & Motivation

This paper deals with communication between Android and iOS platforms via Wi-Fi. Currently, Android is the most commonly used operating system (OS) for smart phones with a market share of 85.2% (see Figure 1). Second most common is the closed source iOS with 13.8%. Figure 1 reveals that other platforms were eliminated over the last years.

The u'smile research center focuses on Android, however, iOS can not be completely neglected. Hence the question of interoperability between Android and iOS arises. The offline scenario (no internet connection) of general identity verification requires communication between two mobile devices in proximity without the need for additional hardware or infrastructure, so no dedicated Wi-Fi access points are used. Hence a peer-to-peer technology such as Blue tooth or Wi-Fi Direct is needed. Then the peers need to do pairing. The setup of the connection between the two devices should impose a minimum of user interactions (ideally fully automated) otherwise the system may be impractical from a usability perspective. Android forces explicit user interaction for Bluetooth paring, but not for Wi-Fi Direct. Hence an Android app could automatically create a communication channel with another Android device that has the same app installed. Another requirement is sufficient transmission

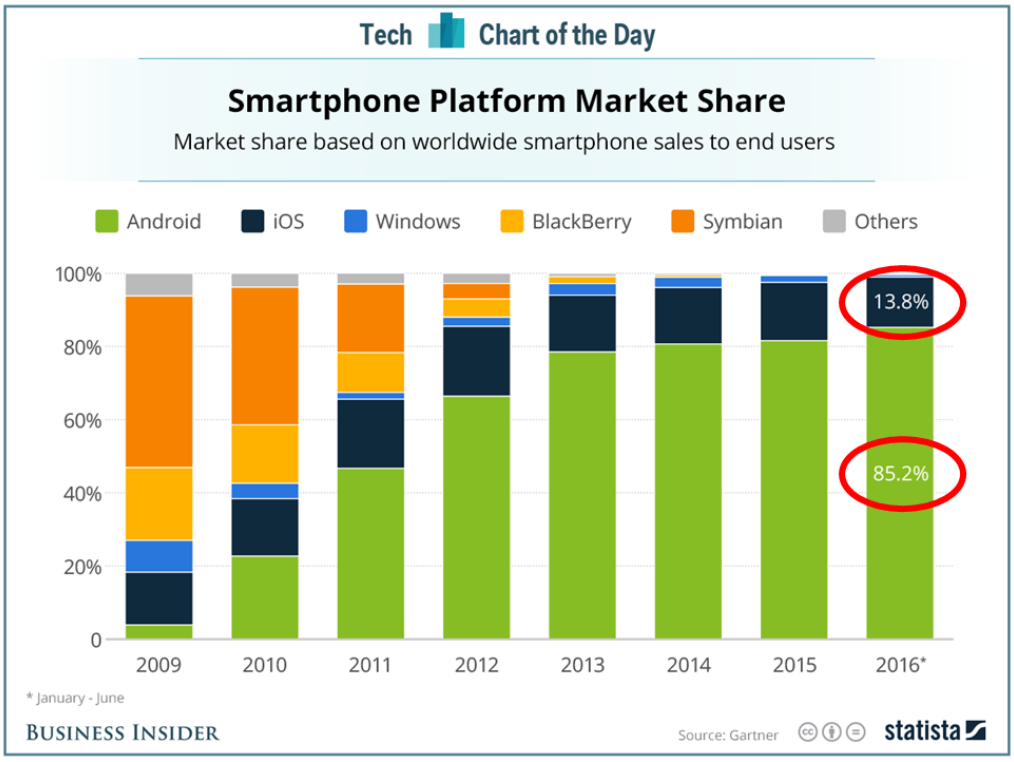


Figure 1: Global Sales Marketshare of smart phones 2009 - 2016

performance, preferably below an user annoyance boundary that we estimate with 1 second. A performance test showed that NFC offers effective transmission speeds of about 2kByte/s. A typical transmission of about 20kB takes 10 seconds. This may be too slow for some use cases. Wi-Fi Direct offers a throughput of about 20 - 40 MByte/s, but needs some time to initialize the connection. The existing Wi-Fi Direct prototype takes about 4 seconds to transfer 20kB, however we see potential for reducing this time by optimizing the Wi-Fi connection setup.

Generally the peers need to support Wi-Fi Direct, however, there is a special *Legacy Mode* that allows a device *with* Wi-Fi Direct support *Android 4.0 or newer* to communicate with a device *without* Wi-Fi Direct support (Apple devices). In that case an Android device becomes a temporary access point and an Apple device (or any other non-Wi-Fi Direct supporting device) could connect to it as if it were a standard infrastructure access point. The Android Wi-Fi Peer-to-Peer API exists for API 14 and above. As a result, currently 97.4% of all Android devices (state Feb 2017) do support Wi-Fi Direct. Therefore, Wi-Fi Direct offers excellent interoperability with other platforms and is widely applicable.

# 2. Wi-Fi Direct Prototype

The Wi-Fi Alliance specifies Wi-Fi Direct [13]. Based on Android Wi-Fi API documentation [10] and a code sample [11], we created an Android prototype that supports peer-to-peer transmission of data between two Android devices. Additionally it is possible to connect an Apple device as legacy client, however, the iOS part is not implemented. The Apple device needs an app installed that handles the connection on the iOS side that would simply connect to an access point and accept a TCP/IP socket. Wi-Fi Direct allows to create networks between multiple devices, however, for simplicity we limit our considerations to communication between two devices.

## 2.1 Potential iOS implementation

Apple implements its own Wi-Fi Direct like protocol called *MultiPeerDiscovery*. It is incompatible with Wi-Fi Direct and uses a combination of Wi-Fi, Bluetooth Classic and Bluetooth Low Energy. MultiPeerDiscovery is closed source and uses combination of Apple's Bonjour for peer discovery and TCP/IP for communication. There is sample code available called WiTab [5] that shows a how to use the peer-to-peer API.

Since November 2016 official Apple documentation exists on how an iOS device could connect to another non-iOS device. The documentation on connecting an iOS device to an Android device via Wi-Fi Direct (Legacy mode) are discussed [7] and

documented [6] iOS development resources. This essentially states that there are several APIs available concerning Wi-Fi networks, but no single general-purpose API for scanning and configuration:

1. *Peer-to-peer networking*: This is the most promissing API for the mobile identity verification use case. The *NSNetService* class represents a network service that may offer the desired functionality.
2. *NEHotspotHelper*: This class allows an app to scan and authenticate to a Wi-Fi hotspot. This class is intended to gain access to the wider Internet, which may be unintended or even problematic in the offline identity verification use case, but of course not in the online use case.
3. *Accessory configuration*: This is intended for connecting gadgets to a mobile device e.g. wireless speakers for playing audio. Here are two APIs available: *Wireless Accessory Configuration* (WAC) and *HomeKit*. Both need to be revised under the Apple's MFI program that closely examines and certifies the gadget device. Android devices are very likely to be denied to participate in that program for organizational and economic reason.

Since these APIs are officially documented, it is allowed to use them and still comply to the iOS app development rules [3] that are checked before an app is made publicly available in the Apple app store. Using undocumented APIs will likely be forbidden if Apple considers it "over the line" [3] which they keep vague and open for interpretation. This could lead to refusal of including the app the store, which prohibits broad adoption for an iOS app.

A simple and effective approach is to create a Wi-Fi profile on the iOS device and delete it again after each session. This is particularly interesting for the combination of Android Wi-Fi Direct in legacy mode and an iOS legacy client. Such a profile is typically created when the device connects to a Wi-Fi network and stores the Service Set Identifier (SSID) and corresponding password so it could automatically connect to it when the known network is in reach again. It is possible to create a Wi-Fi profile from within an app. Wi-Fi profiles are stored in a XML format. Wi-Fi profiles correspond to "Known Groups" in Android Wi-Fi Direct. A similar approach is possible for Android devices that do not yet support Wi-Fi Direct (before Android 4.0).

## 2.2 Wi-Fi Direct Standard Mode

The legancy mode is in the focus for iOS interoperability. However, the standard mode is also shortly summarized for a better understanding of the problems and tasks of Wi-Fi Direct.

The standard mode of Wi-Fi Direct consists of two phases: group formation and communication in an established group. One device acts as access point (group

owner) and the other device as client (group member). During group formation all the participating devices have to do peer discovery to detect each other's presence.

The group owner is determined by an integer between 0 and 15 depending on signal strength, battery charge etc. on each device. If two devices have the same integer a dice role decides the group owner. There is a simple form for both parties agreeing to a protected connection with "Wi-Fi Protected Setup" (WPS) that requires user interaction. However, WPS is *vulnerable* against eavesdrop attackers due to plain text password transmission. In order to circumvent the group owner negotiation and WPS, the Wi-Fi Direct Legacy Mode can be used.

## 2.3 Wi-Fi Direct Legacy Mode

The group owner negotiation can be avoided by directly initiating a group creation, see Android API function `MyWifiP2pManager.creategroup(…)` [9]. WPS transmits the SSID and the password generated by the group owner in *plain text* to the group member. Potentially an eavesdropper could gain access to the SSID and the password. Therefore, another layer of security (e.g. TLS or PACE) is recommendable to ensure confidentiality of the transmission.

The SSID is generated by the group owner (by the Android Wi-Fi Direct implementation). It starts with "DIRECT-" and two time variant pseudo random characters (A-Z or a-z) and ends in additional device specific string (typically the device name), resulting in e.g. "DIRECT-nT-Android_26fa". Therefore collisions of multiple Wi-Fi Direct SSIDs are reasonably unlikely. The SSID is case sensitive.

The Wi-Fi session password (WPA2) is also generated on the group owner side by the Android OS. It is a peudo random string of at least (by default exactly) 8 characters containing upper case and lower case characters (e.g. "zYIJdGFx"). This results in $54^8$ possible passwords.

Both the SSID and the password are generated by the Android OS. They can be queried with the *Wi-FiP2pGroup* class using *getNetworkName()* and *getPassphrase()*. Not only the mobile identity app but also other apps on the device could potentially gain access to this information. The setter functions for the SSID and the password are not visible by default. However, with reflection it is possible for an app to set the SSID and password to arbitrary values to e.g. increase the password strength or to improve the entropy of the SSID, as shown in Listing 1. This can be classified as loophole in the Android Wi-Fi Direct API and is public knowledge [2] since at least 2014.

Listing 1: Set SSID and password through reflection (tested on Android 4.4/API 19)

```
1 public void onGroupInfoAvailable(Wi-FiP2pGroup group) {
2   Method setPassPhraseMethod=group.getClass()
```

```
3         .getMethod("setPassphrase", new Class[] {String.class});
4    setPassPhraseMethod.invoke(group, "yourNewPassPrhase");
5    Method setNetworkName=group.getClass()
6         .getMethod("setNetworkName", new Class[] {String.class});
7    setPassPhraseMethod.invoke(group, "yourNewSSID")
8  ...
9  }
```

The exchange of the SSID and the password could be done through an alternative channel such as NFC or a QR code. This would effectively substitute WPS. Additionally, it is advisable to exchange IP addresses including a (pseudo) random port number. For a group member the IP of the group owner is easily available through the API for the group member, but *not* the other way around. The group member IP could be predefined or transmitted from the group member to the group owner.

## 2.4 BSSID Address Restriction

Restricting access to the wireless network is a sound idea, but faces a practical issue with QR codes. If that restriction is based on the Basic Service Set Identification (BSSID) address[1], this restriction can only be enforced on the soft access point side, which is the group owner in Wi-Fi Direct terminology. However, the group owner is the one who generates SSID and password and has to transmit the data to the group member. The BSSID of the Member needs to be transmitted in the other direction: group member to group owner. Therefore, a bidirectional channel would be required, which may not be practical with QR codes. So we recommend to avoid BSSID restriction and focus on IP address based restriction instead. This restriction may be reused for the online identity verification use case where both participants have public IP addresses. Alternatively to QR codes a technology that already supports bidirectional communication (e.g. NFC) could be employed. With NFC as Out of Band channel (OOB) a BSSID restriction can be practical.

Building a reliable cryptographic trust can only happen at a higher level such as a secure Transport Layer Security (TLS) socket. Also PACE or similar protocols can be used to establish an authenticated channel. One reason why such a protocol is needed is that an established Wi-Fi connection is available for all apps on an Android device. Hence, malicious apps that are already on an Android device may use the established communication channel to interfere with the data exchange. With authenticated protocol (TLS) the dedicated app will be a valid communication partner and malicious apps are locked out. For TLS authentication, certificates have to be exchanged OOB. Furthermore, an attacker might try to eavesdrop on the initial communication. In case of using a QR code, somebody with a high distance

[1]For wire bound communication this is called Media Access Control (MAC) address, an unique ID for a network interface.

lense and camera might get hold of the QR code which contains the SSID and password to connect to the wireless network. Such an attacker would still need to be within the range of that soft access point, which can be approximately 100m depending on the transmission power and the environment. NFC as OOB channel needs close proximity, typical less than 1 cm. For this reason, eavesdropping is significantly harder. Furthermore with NFC an encrypted transmission of the OOB data is possible with e.g. Diffi Hellman and AES. It would require special antennas and expensive receiving electronics to extend the range. An NFC channel is possible between Android devices with NFC support. The downside is that not all Android device support NFC. Apple phones and tablets do *not support* NFC. Therefore, cross platform communication at the moment only possible via unencrypted QR codes as an external channel, given the no user interaction requirement. Encrypted QR codes would require a pre-shared secret (key or certificate) that would introduce Linkability which may raise privacy concerns for identity verification.

## 3. Potential Use Case: Mobile Driving Licence

One particular use case of a Wi-Fi communication channel is a digital driving license verification. The driving license could be stored on a phone or tablet as a substitution or addition of the paper version. It could allow selective digital identity verification. Selective in a sense, that sometimes it is enough to prove to be an adult e.g. 18 years or older. Though, a policeman may be allowed to see and verify the whole diving license data. This use case can further be split into an online scenario where both participants have an Internet connection available and an offline scenario where no Internet connection is available at the time of verification.

If the prover, for example a citizen, tries to prove his identity to the verifier (authority officer), the involved devices need a channel of communication to exchange data. The two devices may be an Android and an iOS device. Assuming the Wi-Fi Direct legacy mode is used for this use case, then some consequences can be derived. Since iOS does not support Wi-Fi Direct, the iOS device can only be the *legacy client* (group member). As a result the second device has to be a device *with* Wi-Fi Direct support (group owner).

In this use case we have to promote compatibility for the citizen role. So the prover can be *either* a legacy or a non-legacy device. As a result, the *verifier always* has to be a Wi-Fi Direct supporting device such as an Android device. We recommend the *Wi-Fi Direct supporting device to start the soft access point* via the according API. Theoretically, also an iOS device could start the soft access point via one of the mentioned iOS APIs, but then Wi-Fi Direct would be used by non of the involved parties, neither prover nor verifier. An Android device could connect to an access point without Wi-Fi Direct support. Hence the functionality provided by Wi-Fi Direct would have to be reinvented as needed, which will likely require system level

permissions. So we choose the *verifier* to be a Wi-Fi Direct *supporting* device, that starts the access point via Wi-Fi Direct and therefore is *group owner*.

The prover user can still be asked in the prover app for explicit consent before releasing any data. However, the time intensive connection setup could already begin automatically after the OOB data (QR code) is transfered. Higher layer protocols (TLS, PACE, etc.) that ensure authenticated communication may need to exchange some parameters such as Card Access Number or PACE password between verifier and prover. Then the OOB data would have to be extended for those parameters.

## 4. Related Projects

- Devicescape [8] is a company that advertises an app that can automatically connect a smart phone to Wi-Fi networks without user interaction in order to show coupons on a customer smart phone within store proximity.
- p2pkit.io [12] sells a library that allows to detect nearby iOS and Android devices and exchange data between each other.
- The AllSeenAlliance [1] is an organization which provides a framework for devices and apps to discover and securely communicate with each other.
- The GoPro sport camera app is available for Android as well as iOS and allows Wi-Fi communication between the platforms. The iOS app is likely part of Apple's MFI Program [4].
- Airdrop is an application by Apple that uses the propitiatory Apple Multi-PeerDiscovery protocol that allows to easily transfer data from one iOS device to another.

## 5. Summary

We created a working prototype that allows two Android devices to transmit data without user interaction via Wi-Fi Direct. For a connection between Android and iOS devices initial information has to be shared using a QR code. There is documentation available for iOS that describes how an iOS device (without Wi-Fi Direct support) could connect to an Android device that has Wi-Fi Direct support via a *legacy mode*. There is no prototype for iOS, however, this document describes how it could be implemented.

# References

[1] Allseen Alliance: Open standards that enable devices, apps and services to discover and connect, https://allseenalliance.org/, Accessed Feb 2017

[2] anime@stackoverflow: How to create an autonomous GO for Wifi Direct in Android with dedicated ssid and passphrase? (2014), http://stackoverflow.com/a/26420601, Accessed Feb 2017

[3] Apple: App Store Review Guidelines, https://developer.apple.com/app-store/review/guidelines/, Accessed Feb 2017

[4] Apple: MFI Program, https://developer.apple.com/programs/mfi/, Accessed Feb 2017

[5] Apple: Sample code WiTab (2014), https://developer.apple.com/library/content/samplecode/WiTap/Introduction/Intro.html

[6] Apple: iOS Wi-Fi Management APIs (2016), https://developer.apple.com/library/content/qa/qa1942/_index.html

[7] Apple Developer Forums: iOS and Wi-Fi Direct (2017), https://forums.developer.apple.com/thread/12885

[8] Devicescape: connectivity and proximity engagement solutions, http://www.devicescape.com, Accessed Feb 2017

[9] Google: Creating P2P Connections with Wi-Fi, https://developer.android.com/training/connect-devices-wirelessly/wifi-direct.html, Accessed Feb 2017

[10] Google: Wi-Fi Peer-to-Peer, https://developer.android.com/guide/topics/connectivity/wifip2p.html, Accessed Feb 2017

[11] Jukka Silvennoinen: Github sample code ”MyWifiMesh” (2015), https://github.com/DrJukka/MyWifiMesh

[12] P2Pkit: Discover, range & exchange with nearby iOS and Android devices, http://p2pkit.io, Accessed Feb 2017

[13] Wi-Fi Alliance: Wi-Fi Multimedia Technical Specification (2012), https://www.wi-fi.org/downloads-registered-guest/Wi-Fi_WMM_Specification_v1.2.0.pdf