SH

Hi i want to know which structure in AOSP 12 will help me get security type of ssid from scan results

In Android Open Source Project (AOSP) 12, you can use the `ScanResult` class to retrieve the security type of an SSID from scan results.

The `ScanResult` class contains various pieces of information about an access point (AP) obtained from a Wi-Fi network scan, including the security type of the network. You can retrieve the security type using the `capabilities` field of the `ScanResult` object, which contains a string that describes the authentication, key management, and encryption schemes used by the network.

To retrieve the security type of an SSID from a scan result, you can use the following code:

java Copy code

```java
WifiManager wifiManager = (WifiManager) getSystemService(Context.WIFI_SERVICE);
List<ScanResult> scanResults = wifiManager.getScanResults();
for (ScanResult scanResult : scanResults) {
    String ssid = scanResult.SSID;
    String capabilities = scanResult.capabilities;
    // Check the security type of the network
    if (capabilities.contains("WPA2")) {
        // The network uses WPA2 security
    } else if (capabilities.contains("WPA")) {
        // The network uses WPA security
    } else if (capabilities.contains("WEP")) {
        // The network uses WEP security
    } else {
        // The network is open
    }
}
```

Regenerate response | Generate PNG | Download PDF | Share Link

which can be one of WPA2, WPA, WEP, or open.

SH are you aware of informationelementutil.java in AOSP framework path

Yes, I am aware of the `InformationElementUtil.java` class in the AOSP framework.

The `InformationElementUtil` class is part of the `android.net.wifi` package and provides utility functions for parsing and encoding information elements in Wi-Fi scan results. Information elements are additional data elements included in beacon and probe response frames in a Wi-Fi network. They can contain information such as the SSID, supported data rates, and the supported authentication and encryption protocols of a network.

The `InformationElementUtil` class provides methods such as `parseInformationElements()` and `buildExtendedCapabilities()`, which can be used to extract and construct information elements in Wi-Fi scan results. These methods can be useful for analyzing the capabilities of Wi-Fi networks and selecting the appropriate network to connect to based on a user's preferences.

Note that the `InformationElementUtil` class is intended for use by the system components and is not part of the public API. Therefore, it may be subject to change in future versions of the Android operating system.

SH ok i want to know i can get security type from InformationElementUtil class for every ssid in wifinative.java

Yes, you can use the `InformationElementUtil` class in combination with the `WifiNative` class to get the security type of every SSID in `wifinative.java`.

The `WifiNative` class is part of the `android.net.wifi` package and provides a JNI interface for interacting with the Wi-Fi driver and firmware. It exposes a number of native methods that can be used to perform Wi-Fi operations such as scanning for

networks, connecting to a network, and retrieving information about the Wi-Fi hardware and driver.

To get the security type of every SSID in `wifinative.java`, you can use the `scan()` method of the `WifiNative` class to perform a Wi-Fi scan and get a list of `ScanResult` objects. Then, for each `ScanResult` object, you can use the `InformationElementUtil` class to extract the information elements and retrieve the security type of the network.

Here's an example code snippet that demonstrates how to get the security type of every SSID in `wifinative.java` using the `InformationElementUtil` class:

java Copy code

```java
import android.net.wifi.ScanResult;
import android.net.wifi.WifiNative;
import android.net.wifi.util.InformationElementUtil;

...

WifiNative wifiNative = new WifiNative();
List<ScanResult> scanResults = wifiNative.scan();

for (ScanResult scanResult : scanResults) {
    String ssid = scanResult.SSID;
    byte[] informationElements = scanResult.informationElements;
    if (informationElements != null) {
        Map<Short, byte[]> ieMap = InformationElementUtil.parseInformationElements(inf
        if (ieMap.containsKey(InformationElementUtil.EID_RSN)) {
            // The network uses WPA2 security
            // You can retrieve other information about the RSN information element us
        } else if (ieMap.containsKey(InformationElementUtil.EID_WPA)) {
            // The network uses WPA security
            // You can retrieve other information about the WPA information element us
        } else if (ieMap.containsKey(InformationElementUtil.EID_PRIVACY)) {
            // The network uses WEP security
        } else {
            // The network is open
        }
    } else {
        // The network is open
```

```
    }
}
```

In this example, `WifiNative` is used to perform a Wi-Fi scan and get a list of `ScanResult` objects. For each `ScanResult` object, the `informationElements` field is checked to see if it contains information elements. If information elements are present, they are parsed using the `InformationElementUtil` class and the security type of the network is determined based on the presence of specific information elements.

SH can you update it to include eap type as well

Yes, I can update the example code snippet to include the EAP type as well.

Here's an updated code snippet that retrieves both the security type and EAP type of each SSID in `wifinative.java`:

scss Copy code

```
import android.net.wifi.ScanResult;
import android.net.wifi.WifiNative;
import android.net.wifi.util.InformationElementUtil;

...

WifiNative wifiNative = new WifiNative();
List<ScanResult> scanResults = wifiNative.scan();

for (ScanResult scanResult : scanResults) {
    String ssid = scanResult.SSID;
    byte[] informationElements = scanResult.informationElements;
    if (informationElements != null) {
        Map<Short, byte[]> ieMap = InformationElementUtil.parseInformationElements(inf
        if (ieMap.containsKey(InformationElementUtil.EID_RSN)) {
            // The network uses WPA2 security
            // You can retrieve other information about the RSN information element us
            byte[] rsnIE = ieMap.get(InformationElementUtil.EID_RSN);
```

```
                int eapType = InformationElementUtil.getEapTypeFromRsnElement(rsnIE);
                if (eapType != InformationElementUtil.EAP_NONE) {
                    // The network uses EAP authentication
                    // You can retrieve the EAP type using the InformationElementUtil meth
                    int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnElemen
                }
            } else if (ieMap.containsKey(InformationElementUtil.EID_WPA)) {
                // The network uses WPA security
                // You can retrieve other information about the WPA information element us
                byte[] wpaIE = ieMap.get(InformationElementUtil.EID_WPA);
                int eapType = InformationElementUtil.getEapTypeFromWpaElement(wpaIE);
                if (eapType != InformationElementUtil.EAP_NONE) {
                    // The network uses EAP authentication
                    // You can retrieve the EAP type using the InformationElementUtil meth
                    int innerEapType = InformationElementUtil.getInnerEapTypeFromWpaElemen
                }
            } else if (ieMap.containsKey(InformationElementUtil.EID_PRIVACY)) {
                // The network uses WEP security
            } else {
                // The network is open
            }
        } else {
            // The network is open
        }
    }
```

In this updated code snippet, the security type of the network is determined based on the presence of the RSN or WPA information elements, as before. If the network uses WPA or WPA2 security, the EAP type is extracted using the **`InformationElementUtil`** methods **`getEapTypeFromRsnElement()`** or **`getEapTypeFromWpaElement()`**, respectively. If the network uses EAP authentication, the inner EAP type is also extracted using the **`getInnerEapTypeFromRsnElement()`** or **`getInnerEapTypeFromWpaElement()`** methods. The retrieved EAP types are integers that can be compared against the constants defined in the **`InformationElementUtil`** class to determine the type of EAP authentication used by the network.

SH

what is differenece between wpa_akm_eap and rsn_akm_eap in InformationElementUtil

In the `InformationElementUtil` class, `wpa_akm_eap` and `rsn_akm_eap` are constants used to identify the Authentication and Key Management (AKM) and Extensible Authentication Protocol (EAP) types for WPA and WPA2 security, respectively.

`wpa_akm_eap` is used to identify the AKM and EAP types for WPA security, while `rsn_akm_eap` is used to identify the AKM and EAP types for WPA2 (or RSN, which stands for Robust Security Network) security.

The `rsn_akm_eap` constant is used in the `InformationElementUtil` methods that deal with RSN information elements, such as `getEapTypeFromRsnElement()` and `getInnerEapTypeFromRsnElement()`. Similarly, the `wpa_akm_eap` constant is used in the `InformationElementUtil` methods that deal with WPA information elements, such as `getEapTypeFromWpaElement()` and `getInnerEapTypeFromWpaElement()`.

Note that WPA is an older security standard that is less secure than WPA2, and its use is not recommended. RSN (WPA2) should be used instead if possible.

SH

ok can you update the function to include these check and add inline comments

Sure, here's an updated version of the function that includes checks for `wpa_akm_eap` and `rsn_akm_eap` and inline comments to explain the code:

csharp Copy code

```
private void printWifiNetworkSecurityTypes() {
    WifiNative wifiNative = new WifiNative();
    List<ScanResult> scanResults = wifiNative.scan();

    for (ScanResult scanResult : scanResults) {
        String ssid = scanResult.SSID;
        byte[] informationElements = scanResult.informationElements;
        if (informationElements != null) {
            // Parse the information elements to retrieve the security and EAP types
            Map<Short, byte[]> ieMap = InformationElementUtil.parseInformationElements
            if (ieMap.containsKey(InformationElementUtil.EID_RSN)) {
```

```
                // The network uses WPA2 security
                // Retrieve the RSN information element and extract the AKM and EAP ty
                byte[] rsnIE = ieMap.get(InformationElementUtil.EID_RSN);
                int akmType = InformationElementUtil.getAkmTypeFromRsnElement(rsnIE);
                int eapType = InformationElementUtil.getEapTypeFromRsnElement(rsnIE);
                if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
                    // The network uses WPA2-Enterprise (EAP) authentication
                    int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnEl
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Enterprise (EAP) EAP
                } else {
                    // The network uses WPA2-Personal (PSK) authentication
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Personal (PSK)");
                }
            } else if (ieMap.containsKey(InformationElementUtil.EID_WPA)) {
                // The network uses WPA security
                // Retrieve the WPA information element and extract the AKM and EAP ty
                byte[] wpaIE = ieMap.get(InformationElementUtil.EID_WPA);
                int akmType = InformationElementUtil.getAkmTypeFromWpaElement(wpaIE);
                int eapType = InformationElementUtil.getEapTypeFromWpaElement(wpaIE);
                if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
                    // The network uses WPA-Enterprise (EAP) authentication
                    int innerEapType = InformationElementUtil.getInnerEapTypeFromWpaEl
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA-Enterprise (EAP) EAP
                } else {
                    // The network uses WPA-Personal (PSK) authentication
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA-Personal (PSK)");
                }
            } else if (ieMap.containsKey(InformationElementUtil.EID_PRIVACY)) {
                // The network uses WEP security
                Log.d(TAG, "SSID: " + ssid + " Security: WEP");
            } else {
                // The network is open (unsecured)
                Log.d(TAG, "SSID: " + ssid + " Security: Open (unsecured)");
            }
        } else {
            // The
```

SH nice please include wpa3 also

Sure, here's an updated version of the function that includes checks for WPA3 security:

csharp Copy code

```
private void printWifiNetworkSecurityTypes() {
    WifiNative wifiNative = new WifiNative();
    List<ScanResult> scanResults = wifiNative.scan();

    for (ScanResult scanResult : scanResults) {
        String ssid = scanResult.SSID;
        byte[] informationElements = scanResult.informationElements;
        if (informationElements != null) {
            // Parse the information elements to retrieve the security and EAP types
            Map<Short, byte[]> ieMap = InformationElementUtil.parseInformationElements
            if (ieMap.containsKey(InformationElementUtil.EID_RSN)) {
                // The network uses WPA2 security
                // Retrieve the RSN information element and extract the AKM and EAP ty
                byte[] rsnIE = ieMap.get(InformationElementUtil.EID_RSN);
                int akmType = InformationElementUtil.getAkmTypeFromRsnElement(rsnIE);
                int eapType = InformationElementUtil.getEapTypeFromRsnElement(rsnIE);
                if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
                    // The network uses WPA2-Enterprise (EAP) authentication
                    int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnEl
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Enterprise (EAP) EAP
                } else {
                    // The network uses WPA2-Personal (PSK) authentication
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Personal (PSK)");
                }
            } else if (ieMap.containsKey(InformationElementUtil.EID_WPA)) {
                // The network uses WPA security
                // Retrieve the WPA information element and extract the AKM and EAP ty
                byte[] wpaIE = ieMap.get(InformationElementUtil.EID_WPA);
                int akmType = InformationElementUtil.getAkmTypeFromWpaElement(wpaIE);
                int eapType = InformationElementUtil.getEapTypeFromWpaElement(wpaIE);
                if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
                    // The network uses WPA-Enterprise (EAP) authentication
                    int innerEapType = InformationElementUtil.getInnerEapTypeFromWpaEl
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA-Enterprise (EAP) EAP
                } else {
                    // The network uses WPA-Personal (PSK) authentication
```

```
                Log.d(TAG, "SSID: " + ssid + " Security: WPA-Personal (PSK)");
            }
        } else if (ieMap.containsKey(InformationElementUtil.EID_OWE)) {
            // The network uses Opportunistic Wireless Encryption (OWE)
            Log.d(TAG, "SSID: " + ssid + " Security: OWE");
        } else if (ieMap.containsKey(InformationElementUtil.EID_SAE)) {
            // The network uses Simultaneous Authentication of Equals (SAE)
            Log.d(TAG, "SSID: " + ssid + " Security: SAE");
        } else if (ieMap
```

SH

hi continue from EID_SAE and complete

Sure, here's the completed function that includes checks for WPA3 security:

csharp Copy code

```
private void printWifiNetworkSecurityTypes() {
    WifiNative wifiNative = new WifiNative();
    List<ScanResult> scanResults = wifiNative.scan();

    for (ScanResult scanResult : scanResults) {
        String ssid = scanResult.SSID;
        byte[] informationElements = scanResult.informationElements;
        if (informationElements != null) {
            // Parse the information elements to retrieve the security and EAP types
            Map<Short, byte[]> ieMap = InformationElementUtil.parseInformationElements
            if (ieMap.containsKey(InformationElementUtil.EID_RSN)) {
                // The network uses WPA2 security
                // Retrieve the RSN information element and extract the AKM and EAP ty
                byte[] rsnIE = ieMap.get(InformationElementUtil.EID_RSN);
                int akmType = InformationElementUtil.getAkmTypeFromRsnElement(rsnIE);
                int eapType = InformationElementUtil.getEapTypeFromRsnElement(rsnIE);
                if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
                    // The network uses WPA2-Enterprise (EAP) authentication
                    int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnEl
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Enterprise (EAP) EAP
                } else {
```

```
                    // The network uses WPA2-Personal (PSK) authentication
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Personal (PSK)");
                }
            } else if (ieMap.containsKey(InformationElementUtil.EID_WPA)) {
                // The network uses WPA security
                // Retrieve the WPA information element and extract the AKM and EAP ty
                byte[] wpaIE = ieMap.get(InformationElementUtil.EID_WPA);
                int akmType = InformationElementUtil.getAkmTypeFromWpaElement(wpaIE);
                int eapType = InformationElementUtil.getEapTypeFromWpaElement(wpaIE);
                if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
                    // The network uses WPA-Enterprise (EAP) authentication
                    int innerEapType = InformationElementUtil.getInnerEapTypeFromWpaEl
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA-Enterprise (EAP) EAP
                } else {
                    // The network uses WPA-Personal (PSK) authentication
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA-Personal (PSK)");
                }
            } else if (ieMap.containsKey(InformationElementUtil.EID_OWE)) {
                // The network uses Opportunistic Wireless Encryption (OWE)
                Log.d(TAG, "SSID: " + ssid + " Security: OWE");
            } else if (ieMap.containsKey(InformationElementUtil.EID_SAE)) {
                // The network uses Simultaneous Authentication of Equals (SAE)
                Log.d(TAG, "SSID: " + ssid + " Security: SAE");
            } else if (ieMap.containsKey(Information
```

SH

hi continue from where you have left

Sure, here's the completed function that includes checks for WPA3 security:

csharp Copy code

```
private void printWifiNetworkSecurityTypes() {
    WifiNative wifiNative = new WifiNative();
    List<ScanResult> scanResults = wifiNative.scan();

    for (ScanResult scanResult : scanResults) {
        String ssid = scanResult.SSID;
        byte[] informationElements = scanResult.informationElements;
```

```
if (informationElements != null) {
    // Parse the information elements to retrieve the security and EAP types
    Map<Short, byte[]> ieMap = InformationElementUtil.parseInformationElements
    if (ieMap.containsKey(InformationElementUtil.EID_RSN)) {
        // The network uses WPA2 security
        // Retrieve the RSN information element and extract the AKM and EAP ty
        byte[] rsnIE = ieMap.get(InformationElementUtil.EID_RSN);
        int akmType = InformationElementUtil.getAkmTypeFromRsnElement(rsnIE);
        int eapType = InformationElementUtil.getEapTypeFromRsnElement(rsnIE);
        if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
            // The network uses WPA2-Enterprise (EAP) authentication
            int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnEl
            Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Enterprise (EAP) EAP
        } else {
            // The network uses WPA2-Personal (PSK) authentication
            Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Personal (PSK)");
        }
    } else if (ieMap.containsKey(InformationElementUtil.EID_WPA)) {
        // The network uses WPA security
        // Retrieve the WPA information element and extract the AKM and EAP ty
        byte[] wpaIE = ieMap.get(InformationElementUtil.EID_WPA);
        int akmType = InformationElementUtil.getAkmTypeFromWpaElement(wpaIE);
        int eapType = InformationElementUtil.getEapTypeFromWpaElement(wpaIE);
        if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
            // The network uses WPA-Enterprise (EAP) authentication
            int innerEapType = InformationElementUtil.getInnerEapTypeFromWpaEl
            Log.d(TAG, "SSID: " + ssid + " Security: WPA-Enterprise (EAP) EAP
        } else {
            // The network uses WPA-Personal (PSK) authentication
            Log.d(TAG, "SSID: " + ssid + " Security: WPA-Personal (PSK)");
        }
    } else if (ieMap.containsKey(InformationElementUtil.EID_OWE)) {
        // The network uses Opportunistic Wireless Encryption (OWE)
        Log.d(TAG, "SSID: " + ssid + " Security: OWE");
    } else if (ieMap.containsKey(InformationElementUtil.EID_SAE)) {
        // The network uses Simultaneous Authentication of Equals (SAE)
        Log.d(TAG, "SSID: " + ssid + " Security: SAE");
    } else if (ieMap.containsKey(Information
```

SH ok can you show me how to check below security type in wifinative.java

SECURITY_EAP_WPA3_ENTERPRISE & SECURITY_SAE

Sure, you can check for `SECURITY_EAP_WPA3_ENTERPRISE` and `SECURITY_SAE` in the `printWifiNetworkSecurityTypes` method as follows:

csharp Copy code

```
private void printWifiNetworkSecurityTypes() {
    WifiNative wifiNative = new WifiNative();
    List<ScanResult> scanResults = wifiNative.scan();

    for (ScanResult scanResult : scanResults) {
        String ssid = scanResult.SSID;
        byte[] informationElements = scanResult.informationElements;
        if (informationElements != null) {
            // Parse the information elements to retrieve the security and EAP types
            Map<Short, byte[]> ieMap = InformationElementUtil.parseInformationElements
            if (ieMap.containsKey(InformationElementUtil.EID_RSN)) {
                // The network uses WPA2 security
                // Retrieve the RSN information element and extract the AKM and EAP ty
                byte[] rsnIE = ieMap.get(InformationElementUtil.EID_RSN);
                int akmType = InformationElementUtil.getAkmTypeFromRsnElement(rsnIE);
                int eapType = InformationElementUtil.getEapTypeFromRsnElement(rsnIE);
                if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
                    // The network uses WPA2-Enterprise (EAP) authentication
                    int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnEl
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Enterprise (EAP) EAP
                } else {
                    // The network uses WPA2-Personal (PSK) authentication
                    Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Personal (PSK)");
                }
            } else if (ieMap.containsKey(InformationElementUtil.EID_WPA)) {
                // The network uses WPA security
                // Retrieve the WPA information element and extract the AKM and EAP ty
                byte[] wpaIE = ieMap.get(InformationElementUtil.EID_WPA);
                int akmType = InformationElementUtil.getAkmTypeFromWpaElement(wpaIE);
                int eapType = InformationElementUtil.getEapTypeFromWpaElement(wpaIE);
                if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
                    // The network uses WPA-Enterprise (EAP) authentication
                    int innerEapType = InformationElementUtil.getInnerEapTypeFromWpaEl
```

```
                Log.d(TAG, "SSID: " + ssid + " Security: WPA-Enterprise (EAP) EAP
            } else {
                // The network uses WPA-Personal (PSK) authentication
                Log.d(TAG, "SSID: " + ssid + " Security: WPA-Personal (PSK)");
            }
        } else if (ieMap.containsKey(InformationElementUtil.EID_OWE)) {
            // The network uses Opportunistic Wireless Encryption (OWE)
            Log.d(TAG, "SSID: " + ssid + " Security: OWE");
        } else if (ieMap.containsKey(InformationElementUtil.EID_SAE)) {
            // The network uses Simultaneous Authentication of Equals (SAE)
            Log.d(TAG, "
```

SH

instead of printWifiNetworkSecurityTypes can you write in new function which can check wpa2 and wpa3 .

Sure, here's an example of a new function called `checkWifiSecurityType` which can check for WPA2, WPA3-Personal, WPA3-Enterprise, OWE and SAE security types:

csharp Copy code

```
private void checkWifiSecurityType(ScanResult scanResult) {
    String ssid = scanResult.SSID;
    byte[] informationElements = scanResult.informationElements;
    if (informationElements != null) {
        // Parse the information elements to retrieve the security and EAP types
        Map<Short, byte[]> ieMap = InformationElementUtil.parseInformationElements(inf
        if (ieMap.containsKey(InformationElementUtil.EID_RSN)) {
            // The network uses WPA2 security
            // Retrieve the RSN information element and extract the AKM and EAP types
            byte[] rsnIE = ieMap.get(InformationElementUtil.EID_RSN);
            int akmType = InformationElementUtil.getAkmTypeFromRsnElement(rsnIE);
            int eapType = InformationElementUtil.getEapTypeFromRsnElement(rsnIE);
            if (akmType == InformationElementUtil.AKM_EAP && eapType != InformationEle
                // The network uses WPA2-Enterprise (EAP) authentication
                int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnElemen
                Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Enterprise (EAP) EAP Typ
            } else {
                // The network uses WPA2-Personal (PSK) authentication
```

```
                Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Personal (PSK)");
            }
        } else if (ieMap.containsKey(InformationElementUtil.EID_WPA)) {
            // The network uses WPA security
            // Retrieve the WPA information element and extract the AKM and EAP types
            byte[] wpaIE = ieMap.get(InformationElementUtil.EID_WPA);
            int akmType = InformationElementUtil.getAkmTypeFromWpaElement(wpaIE);
            int eapType = InformationElementUtil.getEapTypeFromWpaElement(wpaIE);
            if (akmType == InformationElementUtil.AKM_EAP && eapType != InformationEle
                // The network uses WPA-Enterprise (EAP) authentication
                int innerEapType = InformationElementUtil.getInnerEapTypeFromWpaElemen
                Log.d(TAG, "SSID: " + ssid + " Security: WPA-Enterprise (EAP) EAP Type
            } else {
                // The network uses WPA-Personal (PSK) authentication
                Log.d(TAG, "SSID: " + ssid + " Security: WPA-Personal (PSK)");
            }
        } else if (ieMap.containsKey(InformationElementUtil.EID_OWE)) {
            // The network uses Opportunistic Wireless Encryption (OWE)
            Log.d(TAG, "SSID: " + ssid + " Security: OWE");
        } else if (ieMap.containsKey(InformationElementUtil.EID_SAE)) {
            // The network uses Simultaneous Authentication of Equals (SAE)
            Log.d(TAG, "SSID: " + ssid + " Security: SAE");
        } else if (ieMap.containsKey(InformationElement
```

SH

ok i want to check in below order
WPA3-Personal, WPA3-Enterprise, WPA2,SAE

Sure, here's an updated example of the `checkWifiSecurityType` function to check for the security types in the order you specified:

csharp Copy code

```
private void checkWifiSecurityType(ScanResult scanResult) {
    String ssid = scanResult.SSID;
    byte[] informationElements = scanResult.informationElements;
    if (informationElements != null) {
        // Parse the information elements to retrieve the security and EAP types
```

```
Map<Short, byte[]> ieMap = InformationElementUtil.parseInformationElements(inf
if (ieMap.containsKey(InformationElementUtil.EID_OWE)) {
    // The network uses Opportunistic Wireless Encryption (OWE)
    Log.d(TAG, "SSID: " + ssid + " Security: OWE");
} else if (ieMap.containsKey(InformationElementUtil.EID_SAE)) {
    // The network uses Simultaneous Authentication of Equals (SAE)
    Log.d(TAG, "SSID: " + ssid + " Security: SAE");
} else if (ieMap.containsKey(InformationElementUtil.EID_RSN)) {
    // The network uses WPA2 or WPA3 security
    // Retrieve the RSN information element and extract the AKM and EAP types
    byte[] rsnIE = ieMap.get(InformationElementUtil.EID_RSN);
    int akmType = InformationElementUtil.getAkmTypeFromRsnElement(rsnIE);
    int eapType = InformationElementUtil.getEapTypeFromRsnElement(rsnIE);
    if (akmType == InformationElementUtil.AKM_EAP && eapType != InformationEle
        // The network uses WPA3-Enterprise (EAP) authentication
        int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnElemen
        Log.d(TAG, "SSID: " + ssid + " Security: WPA3-Enterprise (EAP) EAP Typ
    } else if (akmType == InformationElementUtil.AKM_SAE) {
        // The network uses WPA3-Personal (SAE) authentication
        Log.d(TAG, "SSID: " + ssid + " Security: WPA3-Personal (SAE)");
    } else {
        // The network uses WPA2 security
        if (akmType == InformationElementUtil.AKM_EAP && eapType != Informatio
            // The network uses WPA2-Enterprise (EAP) authentication
            int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnEl
            Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Enterprise (EAP) EAP
        } else {
            // The network uses WPA2-Personal (PSK) authentication
            Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Personal (PSK)");
        }
    }
} else if (ieMap.containsKey(InformationElementUtil.EID_WPA)) {
    // The network uses WPA security
    // Retrieve the WPA information element and extract the AKM and EAP types
    byte[] wpaIE = ieMap.get(InformationElementUtil.EID_WPA);
    int akmType = InformationElementUtil.getAkmTypeFromWpaElement(wpaIE);
    int eapType = InformationElementUtil.getEapTypeFrom
```

SH can you highlight which line is checking for SECURITY_EAP_WPA3_ENTERPRISE and SECURITY_SAE

Sure, here are the relevant lines of code that check for `SECURITY_EAP_WPA3_ENTERPRISE` and `SECURITY_SAE`:

csharp Copy code

```
if (ieMap.containsKey(InformationElementUtil.EID_SAE)) {
    // The network uses Simultaneous Authentication of Equals (SAE)
    Log.d(TAG, "SSID: " + ssid + " Security: SAE");
} else if (ieMap.containsKey(InformationElementUtil.EID_RSN)) {
    // The network uses WPA2 or WPA3 security
    // Retrieve the RSN information element and extract the AKM and EAP types
    byte[] rsnIE = ieMap.get(InformationElementUtil.EID_RSN);
    int akmType = InformationElementUtil.getAkmTypeFromRsnElement(rsnIE);
    int eapType = InformationElementUtil.getEapTypeFromRsnElement(rsnIE);
    if (akmType == InformationElementUtil.AKM_EAP && eapType != InformationElementUtil
        // The network uses WPA3-Enterprise (EAP) authentication
        int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnElement(rsnIE)
        Log.d(TAG, "SSID: " + ssid + " Security: WPA3-Enterprise (EAP) EAP Type: " + e
    } else if (akmType == InformationElementUtil.AKM_SAE) {
        // The network uses WPA3-Personal (SAE) authentication
        Log.d(TAG, "SSID: " + ssid + " Security: WPA3-Personal (SAE)");
    } else {
        // The network uses WPA2 security
        if (akmType == InformationElementUtil.AKM_EAP && eapType != InformationElement
            // The network uses WPA2-Enterprise (EAP) authentication
            int innerEapType = InformationElementUtil.getInnerEapTypeFromRsnElement(rs
            Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Enterprise (EAP) EAP Type: "
        } else {
            // The network uses WPA2-Personal (PSK) authentication
            Log.d(TAG, "SSID: " + ssid + " Security: WPA2-Personal (PSK)");
        }
    }
}
```

The `EID_SAE` check corresponds to the `SECURITY_SAE` type, and the `akmType == InformationElementUtil.AKM_SAE` check corresponds to the `SECURITY_EAP_WPA3_ENTERPRISE` type (since WPA3-Personal uses SAE as the key management algorithm).

SH

not convinced.
below are different security type right?
SECURITY_EAP_WPA3_ENTERPRISE and SECURITY_SAE

Yes, `SECURITY_EAP_WPA3_ENTERPRISE` and `SECURITY_SAE` correspond to different security types:

- `SECURITY_EAP_WPA3_ENTERPRISE`: This security type corresponds to the WPA3-Enterprise (EAP) authentication method, which uses EAP (Extensible Authentication Protocol) for client authentication and key management. The `akmType` variable is set to `InformationElementUtil.AKM_EAP` and the `eapType` variable is set to a non-zero value.
- `SECURITY_SAE`: This security type corresponds to the WPA3-Personal (SAE) authentication method, which uses Simultaneous Authentication of Equals (SAE) for client authentication and key management. The `ieMap` contains the SAE Information Element (`InformationElementUtil.EID_SAE`), which is used to determine that the network uses SAE.

In the code I provided earlier, the `akmType` check I mentioned (`akmType == InformationElementUtil.AKM_SAE`) corresponds to the `SECURITY_SAE` type, and the check for `akmType == InformationElementUtil.AKM_EAP` with a non-zero `eapType` value corresponds to the `SECURITY_EAP_WPA3_ENTERPRISE` type.

SH

ok then what is this RSN_AKM_SAE

`RSN_AKM_SAE` is a constant defined in the `InformationElementUtil` class in AOSP. It stands for "Robust Security Network - Authentication and Key Management with Simultaneous Authentication of Equals", and is a type of authentication method used in WPA3-Personal (SAE).

In the code I provided earlier, the check for `akmType == InformationElementUtil.AKM_SAE`

is checking for both `AKM_SAE` and `RSN_AKM_SAE` authentication methods, which are equivalent for practical purposes.

SH

ok update

Sure, here's an updated function that checks for the security types in the order you requested:

java Copy code

```java
private static String getWifiNetworkSecurityType(ScanResult scanResult) {
    String capabilities = scanResult.capabilities;
    Map<Integer, byte[]> ieMap = InformationElementUtil.parseInformationElements(scanR
    int akmType = InformationElementUtil.getAkmType(ieMap);
    int eapType = InformationElementUtil.getEapType(ieMap);
    boolean isWpa3Personal = false;
    boolean isWpa3Enterprise = false;
    boolean isWpa2 = false;
    boolean isSae = false;

    if (capabilities.contains("WPA3-Personal") || capabilities.contains("WAPI-PSK")) {
        isWpa3Personal = true;
    } else if (akmType == InformationElementUtil.AKM_EAP && eapType != 0) {
        isWpa3Enterprise = true;
    } else if (capabilities.contains("WPA2")) {
        isWpa2 = true;
    } else if (akmType == InformationElementUtil.AKM_SAE || akmType == InformationElem
        isSae = true;
    }

    if (isWpa3Personal) {
        return "WPA3-Personal";
    } else if (isWpa3Enterprise) {
        return "WPA3-Enterprise";
    } else if (isWpa2) {
        return "WPA2";
    } else if (isSae) {
        return "SAE";
    } else {
```

```
            return "Open";
        }
    }
```

This function first checks for WPA3-Personal and WAPI-PSK, then for WPA3-Enterprise, then for WPA2, and finally for SAE. If none of those security types are detected, it returns "Open".

SH ok now are you aware of from() in informationelementutil.java

Yes, I'm aware of the `from()` method in the `InformationElementUtil` class in AOSP. It is a static factory method that takes a byte array containing the raw binary data of an information element and returns an instance of `InformationElement` class.

The `InformationElement` class is a representation of an information element, which is a variable-length field in a beacon or probe response frame that contains information about the access point or the wireless network. The `InformationElement` class provides methods to access the various fields in the information element, such as the element ID, the length of the element, and the data payload.

The `from()` method in the `InformationElementUtil` class is useful when you need to extract information from the information elements in a beacon or probe response frame.

SH i want to check wifinl80211manager.oemsecuritytype in from()

The `from()` method in the `InformationElementUtil` class is not related to the `WifiNative` class, so it doesn't have any knowledge of the `wifinl80211manager.oemsecuritytype` field.

The `from()` method is used to parse the binary data of an information element and

return an instance of the `InformationElement` class that represents that element. It doesn't have any interaction with the `WifiNative` class or any other Android framework classes.

If you want to check the value of the `wifinl80211manager.oemsecuritytype` field in the `WifiNative` class, you should do that directly in the `WifiNative` class or in a class that has access to the `WifiNative` instance.

SH

ok imagine it had access to wifinative.
now show me how

Assuming you have access to an instance of the `WifiNative` class, you can call the `scan` method to perform a scan and get a list of `ScanResult` objects. Then, for each `ScanResult` object, you can extract the `informationElements` field, which is an array of `ScanResult.InformationElement` objects. Finally, for each `ScanResult.InformationElement` object, you can use the `from()` method in the `InformationElementUtil` class to parse the binary data and get an instance of the `InformationElement` class, and then use the methods in the `InformationElement` class to extract the security type information.

Here's an example code snippet that shows how to extract the security type information for each `ScanResult` object:

java Copy code

```java
WifiNative wifiNative = ...; // create an instance of the WifiNative class

List<ScanResult> scanResults = wifiNative.scan();

for (ScanResult scanResult : scanResults) {
    ScanResult.InformationElement[] informationElements = scanResult.informationElemen

    for (ScanResult.InformationElement informationElement : informationElements) {
        byte[] bytes = informationElement.bytes;

        // Parse the binary data and get an instance of the InformationElement class
        InformationElement ie = InformationElementUtil.from(bytes, 0, bytes.length);
```

```
            // Extract the security type information
            boolean isWpa3Personal = ie.hasRsnAkm(suite -> suite == InformationElement.Wif
            boolean isWpa3Enterprise = ie.hasRsnAkm(suite -> suite == InformationElement.W
            boolean isWpa2 = ie.hasRsnAkm(suite -> suite == InformationElement.WifiAkmSuit
            boolean isSae = ie.hasRsnAkm(suite -> suite == InformationElement.WifiAkmSuite

            // TODO: Do something with the security type information
        }
    }
```

Note that this is just an example, and the exact implementation may vary depending on your specific use case. Also, you may need to add additional error checking and handling to ensure that the code works correctly in all scenarios.

SH

NO ,i want to check in from() method in the InformationElementUtil class for below wifinl80211manager.oemsecuritytype

I see, thank you for clarifying. To access the `wifinl80211manager.oemsecuritytype` field from the `InformationElementUtil.from()` method, you would need to pass a reference to the `WifiNative` object into the `from()` method. Here's an example of how you could modify the `InformationElementUtil.from()` method to check for the `wifinl80211manager.oemsecuritytype` field:

java Copy code

```
public static InformationElement from(byte[] bytes, int offset, int length, WifiNative
    InformationElement ie = new InformationElement();
    ByteBuffer data = ByteBuffer.wrap(bytes, offset, length).order(ByteOrder.LITTLE_EN
    // other code here
    // ...
    // Check for the OEM security type
    int oemSecurityType = -1;
    if (wifiNative != null) {
        oemSecurityType = wifiNative.getOemSecurityType();
    }
    if (oemSecurityType == WifiNative.OEM_SECURITY_TYPE_WPA3_SAE) {
        ie.addRsnAkm(WifiAkmSuite.SAE);
```

```
        } else if (oemSecurityType == WifiNative.OEM_SECURITY_TYPE_WPA3_ENTERPRISE) {
            ie.addRsnAkm(WifiAkmSuite.WPA3_ENTERPRISE);
        } else if (oemSecurityType == WifiNative.OEM_SECURITY_TYPE_WPA3_PERSONAL) {
            ie.addRsnAkm(WifiAkmSuite.WPA3_PERSONAL);
        } else if (oemSecurityType == WifiNative.OEM_SECURITY_TYPE_WPA2) {
            ie.addRsnAkm(WifiAkmSuite.WPA2);
        }
        // other code here
        // ...
        return ie;
    }
```

Note that this assumes that the `WifiNative` object is passed in as a parameter to the `from()` method. If the `WifiNative` object is not accessible in the current scope, you would need to find another way to pass it to the method.

SH

instead of wifiNative.getOemSecurityType imagine below is availabe in from()
wifinl80211manager.oemsecuritytype oemsecurity;

If `wifinl80211manager.oemsecuritytype` is available in the `from()` method, you could replace the `wifiNative.getOemSecurityType()` method call with `oemsecurity.getValue()`, assuming that `oemsecurity` is an instance of the `wifinl80211manager.oemsecuritytype` class. Here's an example of how you could modify the `InformationElementUtil.from()` method to use the `oemsecurity` field:

java Copy code

```
public static InformationElement from(byte[] bytes, int offset, int length, wifinl8021
    InformationElement ie = new InformationElement();
    ByteBuffer data = ByteBuffer.wrap(bytes, offset, length).order(ByteOrder.LITTLE_EN
    // other code here
    // ...
    // Check for the OEM security type
    int oemSecurityType = -1;
    if (oemsecurity != null) {
        oemSecurityType = oemsecurity.getValue();
```

```
        }
        if (oemSecurityType == WifiNative.OEM_SECURITY_TYPE_WPA3_SAE) {
            ie.addRsnAkm(WifiAkmSuite.SAE);
        } else if (oemSecurityType == WifiNative.OEM_SECURITY_TYPE_WPA3_ENTERPRISE) {
            ie.addRsnAkm(WifiAkmSuite.WPA3_ENTERPRISE);
        } else if (oemSecurityType == WifiNative.OEM_SECURITY_TYPE_WPA3_PERSONAL) {
            ie.addRsnAkm(WifiAkmSuite.WPA3_PERSONAL);
        } else if (oemSecurityType == WifiNative.OEM_SECURITY_TYPE_WPA2) {
            ie.addRsnAkm(WifiAkmSuite.WPA2);
        }
        // other code here
        // ...
        return ie;
    }
```

Again, note that this assumes that the `oemsecurity` object is passed in as a parameter to the `from()` method. If the `oemsecurity` object is not accessible in the current scope, you would need to find another way to pass it to the method.

SH ok i want you to check ies and store it to oem and then check for RSN_AKM_SAE

If you want to check the information elements (IEs) of a scan result and store the security type to an `oemsecurity` object, you could modify the `getScanResultSecurity()` method as follows:

java Copy code

```
public static wifinl80211manager.oemsecuritytype getScanResultSecurity(WifiScanner.Sca
    InformationElement[] ies = InformationElementUtil.parseInformationElements(scanRes
    wifinl80211manager.oemsecuritytype oemsecurity = null;
    for (InformationElement ie : ies) {
        if (ie.id == InformationElement.EID_VENDOR_SPECIFIC && ie.bytes != null && ie.
            int oui = ((ie.bytes[0] & 0xFF) << 16) | ((ie.bytes[1] & 0xFF) << 8) | (ie
            if (oui == WifiNative.OUI_TYPE_WFA) {
                if (ie.bytes.length >= 6 && ie.bytes[3] == WifiNative.WFA_KEY_MGMT_OUI
                    int wpaType = ie.bytes[4] & 0xFF;
                    if (wpaType == WifiNative.WFA_KEY_MGMT_SAE) {
```

```
                        return wifinl80211manager.oemsecuritytype.OEM_SECURITY_TYPE_WP
                    } else if (wpaType == WifiNative.WFA_KEY_MGMT_FT_SAE) {
                        return wifinl80211manager.oemsecuritytype.OEM_SECURITY_TYPE_WP
                    }
                }
            } else if (oui == WifiNative.OUI_TYPE_WPA) {
                if (ie.bytes.length >= 4 && ie.bytes[3] == WifiNative.WPA_KEY_MGMT_OUI
                    int wpaType = ie.bytes[4] & 0xFF;
                    if (wpaType == WifiNative.WPA_KEY_MGMT_SAE) {
                        oemsecurity = wifinl80211manager.oemsecuritytype.OEM_SECURITY_
                    } else if (wpaType == WifiNative.WPA_KEY_MGMT_FT_SAE) {
                        oemsecurity = wifinl80211manager.oemsecuritytype.OEM_SECURITY_
                    } else if (wpaType == WifiNative.WPA_KEY_MGMT_PSK) {
                        oemsecurity = wifinl80211manager.oemsecuritytype.OEM_SECURITY_
                    }
                }
            }
        }
    }
    if (oemsecurity != null) {
        return oemsecurity;
    } else {
        return wifinl80211manager.oemsecuritytype.OEM_SECURITY_TYPE_NONE;
    }
}
```

This code should check the IEs of the scan result, and if a matching IE is found for WPA3 SAE or WPA3-Enterprise, the `oemsecurity` object is set to the corresponding value. If no matching IE is found, the method returns `OEM_SECURITY_TYPE_NONE`.

To check for `RSN_AKM_SAE`, you could modify the `from()` method in `InformationElementUtil` as follows:

java Copy code

```
public static InformationElement from(byte[] bytes, int offset, int length, wifinl8021
    InformationElement ie = new InformationElement();
    ByteBuffer data = ByteBuffer.wrap(bytes, offset, length).order(ByteOrder.LITTLE_EN
    // other code here
    // ...
    // Check for the OEM security type
```

```
int
```

SH ok continue from from()

Sure. Here's an updated `from()` method that checks for `RSN_AKM_SAE`:

scss Copy code

```
public static ScanResult.InformationElement[] from(ScanResult scanResult) {
    ScanResult.InformationElement[] ies = new ScanResult.InformationElement[0];
    if (scanResult != null) {
        byte[] ieData = scanResult.informationElements;
        if (ieData != null) {
            try {
                ies = InformationElementUtil.parseInformationElements(ieData);
            } catch (IllegalArgumentException e) {
                Log.e(TAG, "Failed to parse information elements");
            }
        }
    }

    // check for RSN_AKM_SAE in the Information Elements
    for (ScanResult.InformationElement ie : ies) {
        if (ie.id == InformationElement.EID_RSN && ie.bytes != null) {
            Set<Integer> akmSuites = InformationElementUtil.parseAkmSuites(ie.bytes);
            if (akmSuites.contains(AkmSuite.AKM_SAE)) {
                oemsecurity = wifinl80211manager.oemsecuritytype.SAE;
                break;
            }
        }
    }

    return ies;
}
```

In this updated method, the `parseInformationElements` method is called to get the

information elements from the `ScanResult`. Then, the method checks for the presence of `RSN_AKM_SAE` in the Information Elements. If `RSN_AKM_SAE` is present, the `oemsecurity` is set to `wifinl80211manager.oemsecuritytype.SAE`.

SH can you add debug log for all conditions

Sure, here's an updated version of the `from()` method with additional debug logging:

scss Copy code

```
public static ScanResult.InformationElement[] from(ScanResult scanResult) {
    ScanResult.InformationElement[] ies = new ScanResult.InformationElement[0];
    if (scanResult != null) {
        byte[] ieData = scanResult.informationElements;
        if (ieData != null) {
            try {
                ies = InformationElementUtil.parseInformationElements(ieData);
            } catch (IllegalArgumentException e) {
                Log.e(TAG, "Failed to parse information elements");
            }
        }
    }

    Log.d(TAG, "Checking information elements for RSN_AKM_SAE...");
    for (ScanResult.InformationElement ie : ies) {
        if (ie.id == InformationElement.EID_RSN && ie.bytes != null) {
            Set<Integer> akmSuites = InformationElementUtil.parseAkmSuites(ie.bytes);
            if (akmSuites.contains(AkmSuite.AKM_SAE)) {
                Log.d(TAG, "Found RSN_AKM_SAE in information elements");
                oemsecurity = wifinl80211manager.oemsecuritytype.SAE;
                break;
            }
        }
    }

    Log.d(TAG, "No match found for RSN_AKM_SAE in information elements");

    return ies;
```

```
}
```

This version of the method includes additional debug logging for when the method is checking for `RSN_AKM_SAE` in the Information Elements, and whether it has found a match or not.

SH can you list various rsn_akm_..

Sure, here are the various RSN AKM (Authentication and Key Management) suites defined in the IEEE 802.11 standard:

- `RSN_AKM_8021X`: IEEE 802.1X authentication
- `RSN_AKM_PSK`: Pre-Shared Key authentication
- `RSN_AKM_SHA256_8021X`: IEEE 802.1X authentication with SHA-256 hashing
- `RSN_AKM_SHA256_PSK`: Pre-Shared Key authentication with SHA-256 hashing
- `RSN_AKM_FT_8021X`: IEEE 802.1X authentication with Fast Transition
- `RSN_AKM_FT_PSK`: Pre-Shared Key authentication with Fast Transition
- `RSN_AKM_FT_SHA256_8021X`: IEEE 802.1X authentication with Fast Transition and SHA-256 hashing
- `RSN_AKM_FT_SHA384_8021X`: IEEE 802.1X authentication with Fast Transition and SHA-384 hashing
- `RSN_AKM_SAE`: Simultaneous Authentication of Equals

Note that not all of these AKM suites are currently supported on all devices, and support can also depend on the specific firmware or software version running on the device.

SH can you map the result to below

```
public static final int SECURITY_UNKNOWN = -1;
public static final int SECURITY_NONE = 0;
public static final int SECURITY_WEP = 1;
public static final int SECURITY_PSK = 2;
public static final int SECURITY_EAP = 3;
```

```
public static final int SECURITY_OWE = 4;
public static final int SECURITY_SAE = 5;
public static final int SECURITY_EAP_SUITE_B = 6;
public static final int SECURITY_EAP_WPA3_ENTERPRISE = 7;
public static final int SECURITY_WAPI_CERT = 8;
public static final int SECURITY_WAPI_PSK = 9;
public static final int SECURITY_EAP_WPA3_ENTERPRISE_192_BIT = 10;
public static final int SECURITY_PASSPOINT_R1_R2 = 11;
public static final int SECURITY_PASSPOINT_R3 = 12;
public static final int SECURITY_MAX_VAL = 13; // Has to be the last
```

Sure, here's the mapping:

- RSN_AKM_NONE, RSN_AKM_FT_NONE, RSN_AKM_FT_PSK: SECURITY_PSK (2)
- RSN_AKM_PSK, RSN_AKM_PSK_SHA256: SECURITY_PSK (2)
- RSN_AKM_EAP: SECURITY_EAP (3)
- RSN_AKM_EAP_SHA256, RSN_AKM_EAP_SHA384: SECURITY_EAP (3)
- RSN_AKM_FT_EAP, RSN_AKM_FT_EAP_SHA384: SECURITY_EAP (3)
- RSN_AKM_SAE: SECURITY_SAE (5)
- RSN_AKM_EAP_SUITE_B_192, RSN_AKM_EAP_SUITE_B_192_SHA256: SECURITY_EAP_SUITE_B (6)
- RSN_AKM_EAP_SUITE_B_192_SHA384: SECURITY_EAP_SUITE_B (6)
- RSN_AKM_EAP_WPA3_ENTERPRISE: SECURITY_EAP_WPA3_ENTERPRISE (7)
- RSN_AKM_FILS_SHA256, RSN_AKM_FILS_SHA384: SECURITY_EAP (3)
- RSN_AKM_FILS_SHA256_PSK, RSN_AKM_FILS_SHA384_PSK: SECURITY_PSK (2)
- RSN_AKM_EAP_FT, RSN_AKM_EAP_FT_SHA384: SECURITY_EAP (3)
- RSN_AKM_EAP_WPA3_ENTERPRISE_192_BIT: SECURITY_EAP_WPA3_ENTERPRISE_192_BIT (10)

If none of the above AKMs match, then the function can return SECURITY_UNKNOWN (-1).

SH

ok create a table for below mapping the same
02-15 17:58:54.542 1550 2413 I WifiNative: ssid="Ctest_5G_AX3004BCM"scan results capabilities string[WPA2-PSK-CCMP][RSN-PSK+SAE-CCMP][ESS][WPS]
02-15 17:58:54.549 1550 2413 I WifiNative: ssid="dlink_seonbeom.lee"scan results

capabilities string[WPA2-PSK-CCMP+TKIP][RSN-PSK-CCMP+TKIP][WPA-PSK-CCMP+TKIP][ESS][WPS]
02-15 17:58:54.551 1550 2413 I WifiNative: ssid="NETGEAR_hyungmin.shin_2.4G"scan results capabilities string[WPA2-PSK-CCMP][RSN-PSK-CCMP][ESS][WPS]
02-15 17:58:54.552 1550 2413 I WifiNative: ssid="wps_nfc_ap"scan results capabilities string[WPA2-PSK-CCMP][RSN-PSK-CCMP][ESS][WPS]
02-15 17:58:54.553 1550 2413 I WifiNative: ssid="Iptime_kyungjun1.park"scan results capabilities string[WPA2-PSK-CCMP][RSN-PSK-CCMP][ESS]
02-15 17:58:54.554 1550 2413 I WifiNative: ssid="###AutoTest_WPA"scan results capabilities string[WPA-PSK-TKIP][ESS][WPS]
02-15 17:58:54.555 1550 2413 I WifiNative: ssid="Netgear_WNDR3700v3"scan results capabilities string[WPA2-PSK-CCMP+TKIP][RSN-PSK-CCMP+TKIP][WPA-PSK-CCMP+TKIP][ESS]
02-15 17:58:54.555 1550 2413 I WifiNative: ssid="LGE_Welcome"scan results capabilities string[ESS]
02-15 17:58:54.556 1550 2413 I WifiNative: ssid="SDSS_AC1"scan results capabilities string[ESS]
02-15 17:58:54.556 1550 2413 I WifiNative: ssid="ASUS_donghyun13.kim"scan results capabilities string[WPA2-PSK-CCMP][RSN-PSK-CCMP][ESS][WPS]
02-15 17:58:54.556 1550 2413 I WifiNative: ssid="LGDATA5G"scan results capabilities string[WPA2-PSK-CCMP][RSN-PSK-CCMP][ESS][WPS]
02-15 17:58:54.557 1550 2413 I WifiNative: ssid="Ctest_2.4G_AX3004BCM"scan results capabilities string[WPA2-PSK-CCMP][RSN-PSK+SAE-CCMP][ESS][WPS]
02-15 17:58:54.558 1550 2413 I WifiNative: ssid="dlink_seonbeom.lee-5GHz"scan results capabilities string[WPA2-PSK-CCMP+TKIP][RSN-PSK-CCMP+TKIP][WPA-PSK-CCMP+TKIP][ESS][WPS]
02-15 17:58:54.558 1550 2413 I WifiNative: ssid="###AutoTest_WPA2"scan results capabilities string[WPA2-PSK-CCMP][RSN-PSK-CCMP][ESS][WPS]

!

Sure, here's the table mapping the security types to their corresponding integer values:

Capabilities String	Security Type
[WEP]	SECURITY_WEP
[ESS]	SECURITY_NONE
[WPS]	SECURITY_NONE

Capabilities String	Security Type
[WPA-PSK-TKIP]	SECURITY_PSK
[WPA-PSK-CCMP][WPA2-PSK-CCMP][WPA2-PSK-TKIP]	SECURITY_PSK
[WPA3-SAE-CCMP]	SECURITY_SAE
[WPA3-SUITE-B-192]	SECURITY_EAP_SUITE_B
[WAPI-CERT]	SECURITY_WAPI_CERT
[WAPI-PSK]	SECURITY_WAPI_PSK
[RSN-PSK-CCMP+TKIP][WPA-PSK-CCMP+TKIP]	SECURITY_PSK
[RSN-PSK-CCMP][WPA2-PSK-CCMP]	SECURITY_PSK
[RSN-PSK+SAE-CCMP]	SECURITY_SAE
[OWE]	SECURITY_OWE
[WPA3-SUITE-B-192-CCMP]	SECURITY_EAP_WPA3_ENTERPRISE_192_BIT
[WPA3-SUITE-B-192-CCMP+TKIP]	SECURITY_EAP_WPA3_ENTERPRISE_192_BIT
[WPA3-SUITE-B-192-GCM]	SECURITY_EAP_SUITE_B
[WPA3-SUITE-B-192-GCM+TKIP]	SECURITY_EAP_SUITE_B
[WPA3-SUITE-B-384-CCMP]	SECURITY_EAP_SUITE_B
[WPA3-SUITE-B-384-CCMP+TKIP]	SECURITY_EAP_SUITE_B
[WPA3-SUITE-B-384-GCM]	SECURITY_EAP_SUITE_B
[WPA3-SUITE-B-384-GCM+TKIP]	SECURITY_EAP_SUITE_B
[WPA3-SAE-FT-CCMP]	SECURITY_EAP_WPA3_ENTERPRISE
[WPA3-SAE-FT-CCMP+TKIP]	SECURITY_EAP_WPA3_ENTERPRISE
[WPA3-SAE-FT-GCM]	SECURITY_EAP_WPA3_ENTERPRISE
[WPA3-SAE-FT-GCM+TKIP]	SECURITY_EAP_WPA3_ENTERPRISE
[WPA3-SAE-FT-192-CCMP]	SECURITY_EAP_WPA3_ENTERPRISE_192_BIT
[WPA3-SAE-FT-192-CCMP+TKIP]	SECURITY_EAP_WPA3_ENTERPRISE_192_BIT
[WPA3-SAE-FT-192-GCM]	SECURITY_EAP_WPA3_ENTERPRISE
[WPA3-SAE-FT-192-GCM+TKIP]	SECURITY_EAP_WPA3_ENTERPRISE
[HS20]	SECURITY_PASSPOINT_R1_R2
[HS20-HS2]	SECURITY_PASSPOINT_R3


SH

Capabilities String	Security Type
All other cases	SECURITY_UNKNOWN

can you include RSN_AKM .. also

Sure, here is an updated table that includes RSN_AKM values:

Capabilities String	Security Level
[ESS]	SECURITY_NONE
[WEP][ESS]	SECURITY_WEP
[WPA-PSK-CCMP][ESS][WPS]	SECURITY_PSK
[WPA2-PSK-CCMP][ESS][WPS]	SECURITY_PSK
[WPA-PSK-CCMP][RSN-PSK-CCMP][ESS][WPS]	SECURITY_PSK
[WPA2-PSK-CCMP][RSN-PSK-CCMP][ESS][WPS]	SECURITY_PSK
[WPA2-PSK-CCMP][RSN-PSK+SAE-CCMP][ESS][WPS]	SECURITY_SAE
[WPA-PSK-CCMP][RSN-PSK+TKIP][ESS][WPS]	SECURITY_PSK
[WPA2-PSK-CCMP+TKIP][RSN-PSK-CCMP+TKIP][WPA-PSK-CCMP+TKIP][ESS]	SECURITY_PSK
[WPA-PSK-CCMP][RSN-PSK-CCMP][ESS]	SECURITY_PSK
[WPA2-PSK-CCMP][RSN-PSK-CCMP][ESS]	SECURITY_PSK
[WPA-PSK-CCMP][RSN-PSK-CCMP][RSN_AKM_PSK][ESS][WPS]	SECURITY_PSK
[WPA2-PSK-CCMP][RSN-PSK-CCMP][RSN_AKM_PSK][ESS][WPS]	SECURITY_PSK
[WPA2-PSK-CCMP][RSN-PSK+SAE-CCMP][RSN_AKM_SAE][ESS][WPS]	SECURITY_SAE
[WPA2-PSK-CCMP+TKIP][RSN-PSK-CCMP+TKIP][RSN_AKM_PSK][ESS]	SECURITY_PSK
[WPA-PSK-TKIP][ESS][WPS]	SECURITY_PSK
[WPA2-PSK-CCMP][RSN-PSK+SAE-CCMP][ESS][WPS]	SECURITY_SAE
[WPA2-PSK-CCMP][RSN-PSK+SAE-CCMP][RSN_AKM_SAE][ESS][WPS]	SECURITY_SAE
[WPA-PSK-CCMP][RSN-PSK-CCMP][RSN_AKM_EAP][ESS]	SECURITY_EAP
[WPA2-PSK-CCMP][RSN-PSK-CCMP][RSN_AKM_EAP][ESS]	SECURITY_EAP
[WPA-PSK-CCMP][RSN-PSK-CCMP][RSN_AKM_EAP][ESS][WPS]	SECURITY_EAP
[WPA2-PSK-CCMP][RSN-PSK-CCMP][RSN_AKM_EAP][ESS][WPS]	SECURITY_EAP
[WPA2-	

SH

i would like to see RSN_AKM.. in seperate coloumn

Sure, I can modify the table to include a separate column for `RSN_AKM`. Here's the updated table:

SSID	Security Type	RSN AKM(s)
Ctest_5G_AX3004BCM	WPA2-PSK-CCMP	RSN-PSK+SAE-CCMP
dlink_seonbeom.lee	WPA2-PSK-CCMP	RSN-PSK-CCMP+TKIP
NETGEAR_hyungmin.shin_2.4G	WPA2-PSK-CCMP	RSN-PSK-CCMP
wps_nfc_ap	WPA2-PSK-CCMP	RSN-PSK-CCMP
Iptime_kyungjun1.park	WPA2-PSK-CCMP	RSN-PSK-CCMP
###AutoTest_WPA	WPA-PSK-TKIP	None
Netgear_WNDR3700v3	WPA2-PSK-CCMP	RSN-PSK-CCMP+TKIP, RSN-PSK-CCMP, RSN-PSK+SAE-CCMP
LGE_Welcome	None	None
SDSS_AC1	None	None
ASUS_donghyun13.kim	WPA2-PSK-CCMP	RSN-PSK-CCMP
LGDATA5G	WPA2-PSK-CCMP	RSN-PSK-CCMP
Ctest_2.4G_AX3004BCM	WPA2-PSK-CCMP	RSN-PSK+SAE-CCMP
dlink_seonbeom.lee-5GHz	WPA2-PSK-CCMP	RSN-PSK-CCMP+TKIP, RSN-PSK-CCMP, RSN-PSK+SAE-CCMP
###AutoTest_WPA2	WPA2-PSK-CCMP	RSN-PSK-CCMP

SH

ok tell me why Ctest_2.4G_AX3004BCM is having string wpa2 when i have set security for wpa3

It's possible that the access point is configured to support both WPA3 and WPA2 security modes, and the device that's connecting to the access point may only support WPA2.

When a device tries to connect to a Wi-Fi network, it sends a request to the access point with a list of the security modes it supports. The access point then responds with the highest security mode that both the device and access point support. In this case, since the access point supports both WPA3 and WPA2, and the device may only support WPA2, the access point may respond with WPA2 as the chosen security mode.

You can check the configuration of the access point to confirm if it supports both WPA3 and WPA2.

SH

explain & show how group data cipher suite & pairwise cipher suite look .you can use info in the table if necessary

In Wi-Fi security, there are two types of cipher suites used in encryption: group data cipher suite and pairwise cipher suite.

A group data cipher suite is used for encrypting broadcast and multicast frames, while a pairwise cipher suite is used for encrypting unicast frames between a client device and the Wi-Fi access point.

In the table provided, the cipher suite information is included in the "scan results capabilities string" column. The cipher suites are indicated by their corresponding acronyms: CCMP (Counter-Mode/CBC-MAC Protocol), TKIP (Temporal Key Integrity Protocol), SAE (Simultaneous Authentication of Equals), and GCMP (Galois/Counter Mode Protocol).

For example, in the first row of the table, the capabilities string for the network with SSID "Ctest_5G_AX3004BCM" includes the group data cipher suite "CCMP" and pairwise cipher suites "WPA2-PSK-CCMP" and "RSN-PSK+SAE-CCMP". This means that the broadcast and multicast frames for this network are encrypted using the CCMP cipher suite, while the unicast frames are encrypted using either the WPA2-PSK-CCMP or RSN-PSK+SAE-CCMP cipher suite, depending on the supported security protocols of the client device.

Similarly, the other networks in the table have their group data cipher suite and pairwise cipher suite information listed in the same column, indicating the encryption methods used for their broadcast and multicast frames and unicast frames, respectively.

SH pairwise cipher suites "WPA2-PSK-CCMP" and "RSN-PSK+SAE-CCMP"-->really ?

I apologize for the confusion in my previous response. "WPA2-PSK-CCMP" and "RSN-PSK+SAE-CCMP" are actually group cipher suites, not pairwise cipher suites.

Pairwise cipher suites are used to encrypt data traffic between a client and an access point on a per-session basis. They are negotiated during the 4-way handshake and can be different for each session.

Group cipher suites, on the other hand, are used to encrypt broadcast and multicast traffic on the wireless network. They are negotiated during the group key handshake and are the same for all clients on the network.

In the provided table, the pairwise cipher suites are not explicitly listed, but they can be inferred from the capabilities string. For example, "WPA2-PSK-CCMP" implies that the access point supports CCMP (AES) as the pairwise cipher suite for WPA2-PSK (a type of authentication and key management protocol), and "RSN-PSK+SAE-CCMP" implies that the access point supports CCMP as the pairwise cipher suite for RSN (Robust Security Network, also known as WPA3)-PSK and SAE (Simultaneous Authentication of Equals) protocols.

I hope this clears up any confusion.

SH ok summarize neatly if you are accurate about it

Based on the information provided in the table, it appears that the Wi-Fi networks listed have various security settings and capabilities. Most of the networks are configured with WPA2-PSK security, while a few also have support for WPA-PSK or RSN-PSK+SAE security. The cipher suites used for pairwise encryption are mostly CCMP, with a few networks supporting both CCMP and TKIP.

In terms of the group cipher suites, most networks are using CCMP, while a few also support TKIP or both CCMP and TKIP. The RSN_AKM column indicates which Authentication and Key Management (AKM) protocol is being used, with most networks using either PSK or PSK-SAE.

It's worth noting that some of the networks have WPS enabled, which can potentially pose a security risk. Additionally, the network "Ctest_2.4G_AX3004BCM" is showing a string for WPA2-PSK even though WPA3 has been set, which may indicate a configuration issue.

SH ok then show would be correct or expected for network "Ctest_2.4G_AX3004BCM"

Based on the provided table, the network "Ctest_2.4G_AX3004BCM" has the following security capabilities:

- Group data cipher suite: CCMP
- Pairwise cipher suite: RSN-PSK+SAE-CCMP
- RSN IE AKM suites: SAE

Note that the pairwise cipher suite is not "WPA2-PSK-CCMP" as mentioned in an earlier message. Instead, it is "RSN-PSK+SAE-CCMP" which is a combination of RSN (Robust Security Network) and SAE (Simultaneous Authentication of Equals) cipher

suites.

SH Additionally, the network "Ctest_2.4G_AX3004BCM" is showing a string for WPA2-PSK even though WPA3 has been set, which may indicate a configuration issue.-->what was expected?

If WPA3 has been set for the network "Ctest_2.4G_AX3004BCM", we would expect the corresponding string in the log to show "WPA3-PSK" or "RSN-PSK-SAE+CCMP" instead of "WPA2-PSK-CCMP" or "RSN-PSK+SAE-CCMP". The fact that the log is still showing "WPA2-PSK-CCMP" or "RSN-PSK+SAE-CCMP" for this network may indicate that there is a configuration issue with the network settings.

SH are you aware of generatewpa2capabilitiesstring()

I am not aware of the `generatewpa2capabilitiesstring()` function specifically. It is possible that this function is part of a larger program or script, or it may be a custom function created by the developer who wrote the code you are working with. Can you provide more context about where you encountered this function?

SH AOSP 12 informationelementutil

InformationElementUtil is a class in the Android Open Source Project (AOSP) that provides utility methods for processing Wi-Fi Information Elements (IEs) on Android devices running Android 12 or later. IEs are used to convey additional information about a Wi-Fi network, such as the supported data rates, security capabilities, and vendor-specific information.

InformationElementUtil provides methods for parsing IEs and extracting information such as the capabilities and supported features of a network. It also provides

methods for creating and serializing IEs, which is useful for constructing test cases and generating debug information.

Some of the methods provided by InformationElementUtil include:

- `parseInformationElements(byte[] bytes)`: Parses the specified byte array into a list of Wi-Fi Information Elements.
- `getInformationElement(List<InformationElement> ies, int id)`: Returns the Wi-Fi Information Element with the specified ID from the specified list of IEs.
- `getBssLoadElement(List<InformationElement> ies)`: Returns the BSS Load Information Element from the specified list of IEs.
- `generateWpa2CapabilitiesString(boolean preSharedKey, boolean pmf)`: Generates a string representing the WPA2 capabilities of a Wi-Fi network, based on the specified parameters.
- `createInterworkingElement(int accessNetworkType)`: Creates an Interworking Information Element with the specified access network type.

These are just a few examples of the methods provided by InformationElementUtil. It is a useful tool for developers working on Wi-Fi-related features on Android devices running Android 12 or later.

SH

ok explain me what is wpa2/wpa3 transition mode and how SAE capability is handled during string construction

WPA2/WPA3 Transition mode is a mechanism that allows devices with support for both WPA2 and WPA3 to seamlessly transition between them. In this mode, the Access Point (AP) advertises both WPA2 and WPA3 capabilities, and the client device decides which one to use based on its own capabilities and the security settings configured on the AP.

When constructing the WPA2/WPA3 capabilities string, the SAE (Simultaneous Authentication of Equals) capability is included as part of the WPA3 capability. SAE is a secure key establishment protocol that is used in WPA3-Personal mode.

The format of the capabilities string includes a list of supported security protocols,

key management protocols, and pairwise and group cipher suites. In the case of WPA3, the SAE key management protocol is used to negotiate the initial key, and the CCMP-128 pairwise cipher suite is used for data encryption. The WPA2/WPA3 Transition mode allows clients with WPA3 capability to use SAE for authentication and encryption, while clients with only WPA2 capability will fall back to using 4-way handshake for authentication and TKIP/CCMP cipher suites for encryption.

During the construction of the capabilities string, the SAE capability is included as part of the WPA3 capability, and the string is constructed based on the configured security settings of the AP. If the AP is configured to use WPA3-Personal with SAE, then the SAE capability will be advertised as part of the WPA3 capability in the string. If the AP is configured for WPA2 or WPA3-Enterprise, then the SAE capability will not be included in the capabilities string.

SH

oh i see
now lets focus on MLO attributes.

Sure, I'd be happy to help you with MLO attributes. What would you like to know about them?

SH

first explain me what it is

MLO stands for "Minimum Length Override". It is a feature in Wi-Fi networks that allows the specification of a minimum length for the frames used in the wireless communication. This feature is used to ensure that frames are not fragmented, which can result in reduced network performance.

The MLO attribute is a part of the Wi-Fi configuration and is usually set by the network administrator. It specifies the minimum length of the frames used in the network, and can be set to a value between 256 and 7936 bytes.

When a Wi-Fi device joins the network, it will read the MLO attribute from the

network configuration and use this value to ensure that all frames it sends are of sufficient length. If a frame is too short, the device will pad the frame with additional data to meet the specified minimum length. This padding can be either zeros or random data, depending on the device configuration.

The MLO feature can improve network performance by reducing the number of fragmented frames, which can result in reduced throughput and increased latency. However, it can also increase the overhead of wireless communication, as devices may need to pad frames with additional data to meet the specified minimum length.

SH list them

Here are some examples of MLO attributes:

1. location (e.g., GPS coordinates, address, city, country)
2. size (e.g., square footage, number of rooms, acreage)
3. age (e.g., year built, age of appliances)
4. condition (e.g., new, refurbished, needs repairs)
5. features (e.g., swimming pool, fireplace, hardwood floors)
6. price (e.g., asking price, monthly rent, price per square foot)
7. availability (e.g., move-in date, length of lease)
8. amenities (e.g., parking, laundry facilities, fitness center)
9. pet policy (e.g., allowed, not allowed, additional fee)
10. smoking policy (e.g., allowed, not allowed)

SH hey boss list the mlo attributes related to wifi.ex:-apmlolinkid

Sure, here are some MLO attributes related to Wi-Fi:

- apmlolinkid
- wifimlour
- wifimlodc
- wifimlomode
- wifimlopktsize
- wifimlopkttype
- wifimlostream
- wifimlotxrate
- wifimlotxpower
- wifimlotxstream
- wifimlorxrate
- wifimlorxstream
- wifimlortt
- wifimloperfrssi
- wifimloperfsnr
- wifimloperfbssid
- wifimloperfchan

SH

Note that the specific set of MLO attributes available and their meanings may vary depending on the device and the implementation.
explain and show me how they look

Here are some MLO attributes related to WiFi:

1. `apMloLinkId`: This attribute is used to store the AP's MLO link ID.
2. `apMloPriority`: This attribute is used to store the AP's MLO priority.
3. `wpsCfgMethod`: This attribute is used to store the WPS configuration method.
4. `rsnCfgEnabled`: This attribute is used to store the RSN configuration status.
5. `wpsCfgEnabled`: This attribute is used to store the WPS configuration status.
6. `wpsCfgMode`: This attribute is used to store the WPS configuration mode.
7. `wpsCfgSetup`: This attribute is used to store the WPS configuration setup type.
8. `wpsCfgRegistrar`: This attribute is used to store the WPS configuration registrar.
9. `wpsCfgState`: This attribute is used to store the WPS configuration state.

These attributes can be accessed and modified using the Android WiFi API. Here is an example of how to get the value of the `apMloLinkId` attribute:

java Copy code

```
WifiConfiguration config = new WifiConfiguration();
int apMloLinkId = config.apMloLinkId;
```

Similarly, here is an example of how to set the value of the `apMloPriority` attribute:

java Copy code

```
WifiConfiguration config = new WifiConfiguration();
config.apMloPriority = 5;
```

SH explain what they when we set it

Sure, here's an explanation of what happens when we set some of the MLO attributes related to WiFi:

- `apmlochannel`: This attribute sets the channel on which the access point will operate. Setting this attribute will force the access point to use the specified channel for communication with clients.
- `apmlossid`: This attribute sets the Service Set Identifier (SSID) of the access point. The SSID is a unique identifier that allows clients to connect to the correct access point. Setting this attribute will change the name of the access point as seen by clients.
- `apmloauthmode`: This attribute sets the authentication mode used by the access point. Authentication mode determines how clients are verified before they are granted access to the network. Setting this attribute will change the authentication mode of the access point.
- `apmlocryptotype`: This attribute sets the encryption type used by the access point. Encryption is used to protect the data that is transmitted over the network from being intercepted by unauthorized parties. Setting this attribute will change the encryption type used by the access point.
- `apmlohidden`: This attribute determines whether the access point should

broadcast its SSID or not. Setting this attribute to true will cause the access point to stop broadcasting its SSID, making it invisible to clients that are not already connected to it.

When we set any of these attributes, we are effectively changing the configuration of the access point. For example, if we set `apmlochannel` to 6, the access point will operate on channel 6 for all subsequent communications. Similarly, if we set `apmloauthmode` to "WPA-PSK", the access point will use WPA-PSK authentication for all subsequent connections.

SH

ok tell me for link id

The `apmlolinkid` MLO attribute is used to specify the Link ID of the MLME associated with an AP, which is used to communicate with a STA.

When set, this attribute specifies the Link ID value to use for communication with STAs associated with this AP. The Link ID is a unique identifier that is used to distinguish between multiple connections between a STA and an AP.

For example, in a scenario where a single STA is associated with two different APs, each AP would have a different Link ID value associated with that STA, so that the STA can distinguish between the two connections.

The value of the `apmlolinkid` attribute is a hexadecimal string of 16 bytes, which corresponds to the Link ID value.

SH

oh i see.next explain about radiochaininfos

`RadioChainInfos` is an MLO (Multi Layer Optimization) attribute in Wi-Fi that provides information about the radio chains available on a particular AP (Access Point) or client device. Radio chains are physical antennas that are used to transmit and receive Wi-Fi signals.

The `RadioChainInfos` MLO attribute provides a list of available radio chains and their signal quality (RSSI) in dBm. It also includes information about the channel bandwidth and center frequency of each radio chain.

When this attribute is set for an AP, it allows a client device to make an informed decision about which radio chain to use for optimal performance. The client device can use the information provided by `RadioChainInfos` to select the best available radio chain based on signal quality and other factors.

In addition, when `RadioChainInfos` is set for a client device, it provides the AP with information about the client's radio capabilities, which can help the AP to select the best radio chain to use for communication with the client.

Here is an example of how `RadioChainInfos` might look:

yaml Copy code

```
RadioChainInfos : 1
  RadioChainInfo : 2.4 GHz, BW:20, CF:2437, RSSI:-60
  RadioChainInfo : 5 GHz, BW:80, CF:5180, RSSI:-70
```

In this example, the `RadioChainInfos` attribute indicates that there is one radio chain available. The first `RadioChainInfo` entry provides information about the 2.4 GHz radio chain, which has a 20 MHz bandwidth and a center frequency of 2437 MHz. The RSSI of this radio chain is -60 dBm. The second `RadioChainInfo` entry provides information about the 5 GHz radio chain, which has an 80 MHz bandwidth and a center frequency of 5180 MHz. The RSSI of this radio chain is -70 dBm.

SH RadioChainInfos MLO attribute provides a list of available radio chains and their signal quality (RSSI) in dBm.-->are you sure it is a mlo

I apologize for the confusion. `RadioChainInfos` is not a MLO attribute but a structure used to represent a list of available radio chains and their signal quality in dBm. This structure is used in various parts of the Wi-Fi framework to provide information on

the available radio chains and their signal quality to the system.

SH ok show me a sample config with all the things i discussed so far

Certainly! Here's an example configuration with the MLO attributes and WiFi configuration options we've discussed:

```php
<config>
    <MLO>
        <LinkID>123456</LinkID>
        <RadioChainInfos>
            <RadioChainInfo>
                <ChainID>1</ChainID>
                <RSSI>-75</RSSI>
            </RadioChainInfo>
            <RadioChainInfo>
                <ChainID>2</ChainID>
                <RSSI>-80</RSSI>
            </RadioChainInfo>
        </RadioChainInfos>
    </MLO>
    <Wifi>
        <SSID>MyWiFiNetwork</SSID>
        <Security>SAE</Security>
        <Passphrase>MyPassphrase</Passphrase>
        <TransitionMode>TRUE</TransitionMode>
        <PairwiseCipher>CCMP</PairwiseCipher>
        <GroupCipher>CCMP</GroupCipher>
        <Wpa3Capabilities>CCMP SAE</Wpa3Capabilities>
    </Wifi>
</config>
```

This configuration sets the LinkID to `123456` and provides information about two available radio chains with IDs `1` and `2`. The WiFi network is configured with the SSID `MyWiFiNetwork`, security set to `SAE`, passphrase `MyPassphrase`, and WPA3

capabilities `CCMP SAE`. The `TransitionMode` is set to `TRUE` to enable the WPA2/WPA3 transition mode. The `PairwiseCipher` and `GroupCipher` are set to `CCMP`.

SH

oh ok.
now are you aware of addaccesspoint() in wifinative.java

As a language model, I don't have access to specific implementations or code. However, the method `addAccessPoint()` in `WifiNative.java` is likely a function that adds a new access point to the Wi-Fi network. It may take in parameters such as the SSID, security type, password, and other relevant settings to configure the access point. The exact implementation and usage may depend on the specific application or project using the method.

SH

ok how about softaphalcallbackfromwificond

I apologize, but I am not familiar with "softaphalcallbackfromwificond." Could you provide more context or information about what it is?

SH

ok but answer me first are you aware of startsoftap function in aosp12 framework

Yes, I am aware of the `startSoftAp()` function in AOSP 12 framework. It is a public method in the `WifiNative` class that starts a Soft AP with the given configuration. The configuration includes the SSID, the security parameters (e.g. WPA2, WPA3), the channel, the maximum number of clients, and other parameters that define the Soft AP's behavior.

SH

ok then tell me about softaphalcallbackfromwificond

`SoftApHalCallbackFromWificond` is a callback interface in the Android Wi-Fi framework that is used by `wificond` to inform the framework about changes in the Soft AP state. This interface is implemented in the `WifiServiceImpl` class, which is part of the Android framework.

The `SoftApHalCallbackFromWificond` interface defines several methods that are called by `wificond` when Soft AP events occur. These methods include:

- `onNumAssociatedStationsChanged()`: This method is called when the number of clients associated with the Soft AP changes.
- `onSoftApChannelSwitched()`: This method is called when the Soft AP channel is switched.
- `onSoftApInfoChanged()`: This method is called when any information related to the Soft AP changes.
- `onStaConnected()`: This method is called when a station connects to the Soft AP.
- `onStaDisconnected()`: This method is called when a station disconnects from the Soft AP.

The `SoftApHalCallbackFromWificond` interface is used to receive events from the `wificond` daemon, which is responsible for managing the Wi-Fi hardware on Android devices. When an event occurs, `wificond` notifies the framework through this interface so that the framework can update its internal state accordingly.

SH

i see then next tell me about addAccessPoint()

The `addAccessPoint()` method is a function defined in the `WifiNative` class of the AOSP (Android Open Source Project) framework. This method is responsible for adding a new access point configuration to the Wi-Fi subsystem.

The `addAccessPoint()` method takes a single argument, which is a `Wificonfiguration` object that represents the access point configuration. The `Wificonfiguration` object contains all the necessary parameters required to configure the access point, such as

SSID, security type, authentication type, encryption type, and other network parameters.

When the `addAccessPoint()` method is called, the Wi-Fi subsystem validates the `WifiConfiguration` object and creates a new access point configuration. If the configuration is valid, the access point is added to the Wi-Fi subsystem, and the Wi-Fi driver starts broadcasting the network.

The `addAccessPoint()` method is typically called by the Android framework when a user creates a new Wi-Fi hotspot, and the user interface needs to add the hotspot configuration to the Wi-Fi subsystem. It can also be used by other applications that need to configure an access point programmatically.

Here is an example of how to use the `addAccessPoint()` method to create a new access point configuration:

java Copy code

```
WifiConfiguration config = new WifiConfiguration();
config.SSID = "MyHotspot";
config.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA2_PSK);
config.preSharedKey = "MyPassword";
WifiNative wifiNative = new WifiNative();
wifiNative.addAccessPoint(config);
```

In this example, we create a new `WifiConfiguration` object with the SSID set to "MyHotspot", and the security type set to WPA2-PSK. We also set the pre-shared key to "MyPassword". Finally, we create a new `WifiNative` object and call the `addAccessPoint()` method with the `WifiConfiguration` object as a parameter. This creates a new access point configuration and adds it to the Wi-Fi subsystem.

SH context:-this function is called in startsoftap function after registering for callback from wificond

`addAccessPoint()` is a method in the `WifiNative` class in the AOSP framework that is used to configure and add a Soft AP to the system. It takes in an instance of the

`WifiConfiguration` class, which contains the necessary parameters to configure the Soft AP, such as the SSID, security type, passphrase, etc.

When `addAccessPoint()` is called, it first checks to see if the Soft AP has already been added to the system. If it has, the method returns. If not, it sets the Soft AP configuration using the parameters in the `WifiConfiguration` object and adds it to the system.

After the Soft AP has been added, `startSoftAp()` method uses the `WifiManager` API to enable the Soft AP and start broadcasting the SSID.

It's worth noting that `addAccessPoint()` is a lower-level function used by the `startSoftAp()` method to add the Soft AP to the system. In general, it is not meant to be called directly from an application, but rather is used by system-level components to configure the Soft AP.

SH

ok i this function i recall a inline comment as below
//AP + AP UP

The inline comment `//AP + AP UP` in the `addAccessPoint` function likely refers to the fact that this function is used to both create a new access point and bring up an existing access point. In other words, it handles both the "AP" (access point) creation and the "AP UP" (access point up) process.

When the function is called, it checks whether an existing access point with the same SSID (network name) already exists. If it does, the function will update the existing access point's configuration with the new settings (such as the security mode or channel). If not, the function will create a new access point with the specified configuration.

Overall, the `addAccessPoint` function is a key part of the process for creating and managing access points in the Android Wi-Fi framework.

SH

what is bridge mode in softap

In SoftAP, the "bridge" mode refers to the configuration where the SoftAP creates a bridge between its wireless network interface and its wired network interface. This allows devices connected to the SoftAP's wireless network to communicate directly with devices connected to its wired network interface, as if they were connected to the same physical network.

This can be useful in scenarios where the SoftAP needs to provide

SH

hmm what is setsapsarpowerbackoff

The `setSapSarPowerBackoff` function is used to set the SAR (Specific Absorption Rate) power backoff level for the SoftAP.

SAR is a measure of the rate at which energy is absorbed by the human body when exposed to radio frequency electromagnetic fields, such as those emitted by wireless devices. The SAR power backoff level reduces the transmit power of the SoftAP to reduce the SAR value and maintain compliance with regulatory requirements.

The function takes an integer value representing the power backoff level in dBm, and returns a boolean indicating whether the operation was successful or not.

SH

ok explain the various levels

In the context of the `setsapsarpowerbackoff` function, the "levels" refer to the different possible settings for the SAR (Specific Absorption Rate) power backoff feature. This feature reduces the transmit power of the SoftAP to ensure that the SAR limits are not exceeded, which is important for ensuring user safety.

The SAR power backoff feature can be adjusted to different levels depending on the

regulatory domain and other factors. The specific levels may vary depending on the device and the region, but generally they range from 0 to 15.

Each level corresponds to a specific SAR power backoff value, which is expressed as a percentage of the maximum transmit power. For example, a level of 0 might correspond to no power backoff, while a level of 15 might correspond to a 50% reduction in transmit power.

The appropriate SAR power backoff level for a given device and region is usually determined by regulatory agencies such as the FCC in the United States or the EU's Radio Equipment Directive. The level can also be adjusted based on other factors such as the antenna type, the distance between the device and the user's body, and the specific use case.

SH ok i did not know can you tell more what antenna type

In wireless networking, an antenna is a device that transmits and/or receives electromagnetic waves. It is used to convert electrical signals into radio waves that can be transmitted through the air and vice versa. Antennas come in various types and shapes, each designed for a specific purpose or application.

For example, some common types of antennas used in Wi-Fi networking include:

1. Dipole Antenna: A simple antenna consisting of two metal rods or wires that are placed in a straight line. This type of antenna is commonly used in routers and access points.
2. Omnidirectional Antenna: A type of antenna that radiates signals in all directions. This type of antenna is used when the coverage area needs to be wide and uniform.
3. Directional Antenna: A type of antenna that radiates signals in a particular direction. This type of antenna is used when the coverage area needs to be limited and specific, such as in point-to-point connections.
4. Yagi Antenna: A directional antenna with a more complex design that provides higher gain and more directional coverage.

The choice of antenna type depends on various factors, such as the required coverage area, the desired signal strength, the frequency band, and the environmental conditions.

SH what type of antenna is used in mobile phones

Mobile phones typically use a type of antenna called a PIFA (Planar Inverted-F Antenna). PIFAs are small, flat antennas that are printed onto a circuit board, and are commonly used in smartphones due to their compact size and performance. They are designed to be omnidirectional, meaning they can transmit and receive signals in all directions.

SH oh ok can you show gate level design

As an AI language model, I don't have the ability to create visualizations of gate-level designs. However, I can explain that a gate-level design is a representation of a digital circuit in which the circuit is described in terms of its logical functions, represented by logic gates, and the interconnections between those gates. This representation is typically used in digital circuit design and can be used for verification, optimization, and implementation of the circuit.

A gate-level design can be generated from a higher-level description of the circuit, such as a register transfer level (RTL) description, through a process called synthesis. During synthesis, the high-level description is transformed into a gate-level description that is composed of basic building blocks such as AND, OR, and NOT gates, and their associated logic functions. This gate-level description can then be further optimized, mapped to physical components, and eventually implemented as a digital circuit.

SH hi

Hello! How can I assist you today?

SH

i managed to get lower layer logs for our AP.can you decode and map the info with table

02-09 16:01:08.673 4727 4727 D wpa_supplicant: * SSID=Ctest_2.4G_AX3004BCM
02-09 16:01:08.673 4727 4727 D wpa_supplicant: * IEs - hexdump(len=89): 30 2a 01 00 00 0f ac 04 01 00 00 0f ac 04 01 00 00 0f ac 08 80 00 01 00 f6 cf 24 dd 64 6d c4 dd ...
02-09 16:01:08.673 4727 4727 D wpa_supplicant: * WPA Versions 0x2
02-09 16:01:08.673 4727 4727 D wpa_supplicant: * pairwise=0xfac04
02-09 16:01:08.673 4727 4727 D wpa_supplicant: * group=0xfac04
02-09 16:01:08.673 4727 4727 D wpa_supplicant: * akm=0xfac08
02-09 16:01:08.673 4727 4727 D wpa_supplicant: * Auth Type 0

Sure! Here's a breakdown of the information in the log:

- SSID: "Ctest_2.4G_AX3004BCM"
- IEs - hexdump: This is a dump of information elements (IEs) in hexadecimal format, which are used to convey information between devices in a wireless network. The length of the dump is 89 bytes.
- WPA Versions: This field indicates the versions of WPA supported by the access point. In this case, it indicates that the access point supports WPA version 2.
- Pairwise: This field specifies the pairwise encryption ciphers used for the data traffic between the client and the access point. The value "0xfac04" indicates that the access point supports the following ciphers: CCMP, TKIP, WRAP, and BIP.
- Group: This field specifies the group encryption ciphers used for broadcast/multicast traffic between the client and the access point. The value "0xfac04" indicates that the access point supports the following ciphers: CCMP, TKIP, WRAP, and BIP.
- AKM: This field specifies the authentication and key management protocols (AKMs) supported by the access point. The value "0xfac08" indicates that the access point supports the Simultaneous Authentication of Equals (SAE) protocol.

SH

so akm is wpa3 but wpa version is 2 ?

Yes, it appears that the AKM (Authentication and Key Management) being used is WPA3 (0xfac08), but the WPA version is set to 0x2. This is likely because the WPA version is being used as a fallback in case a client does not support WPA3.