Method 1:

Method:

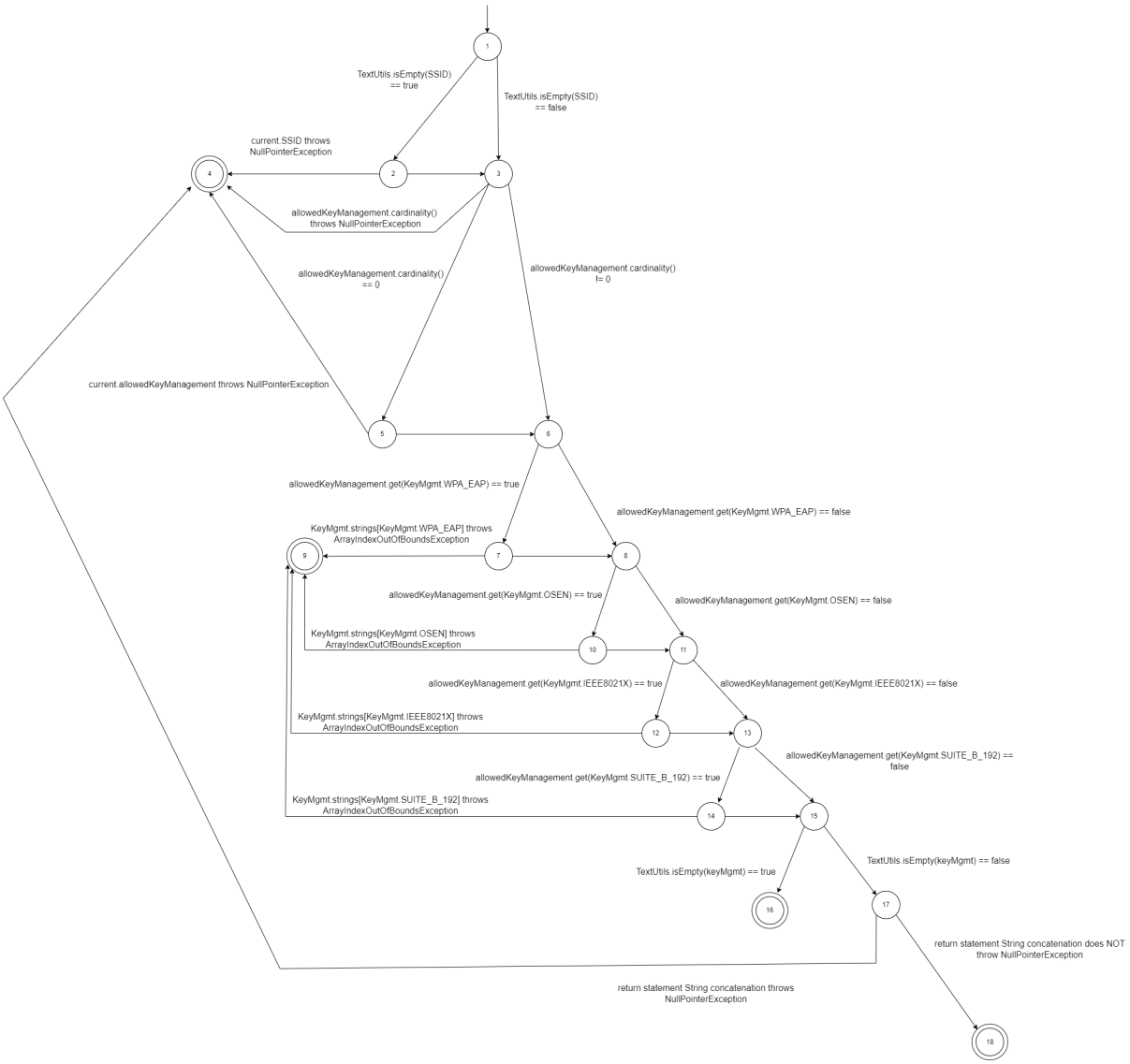
```
public String getKeyIdForCredentials(WifiConfiguration current) {
    String keyMgmt = "";

    try {
        // Get current config details for fields that are not initialized
        if (TextUtils.isEmpty(SSID)) SSID = current.SSID;
        if (allowedKeyManagement.cardinality() == 0) {
            allowedKeyManagement = current.allowedKeyManagement;
        }
        if (allowedKeyManagement.get(KeyMgmt.WPA_EAP)) {
            keyMgmt += KeyMgmt.strings[KeyMgmt.WPA_EAP];
        }
        if (allowedKeyManagement.get(KeyMgmt.OSEN)) {
            keyMgmt += KeyMgmt.strings[KeyMgmt.OSEN];
        }
        if (allowedKeyManagement.get(KeyMgmt.IEEE8021X)) {
            keyMgmt += KeyMgmt.strings[KeyMgmt.IEEE8021X];
        }
        if (allowedKeyManagement.get(KeyMgmt.SUITE_B_192)) {
            keyMgmt += KeyMgmt.strings[KeyMgmt.SUITE_B_192];
        }

        if (TextUtils.isEmpty(keyMgmt)) {
            throw new IllegalStateException("Not an EAP network");
        }

        return trimStringForKeyId(SSID) + "_" + keyMgmt + "_" +
                trimStringForKeyId(enterpriseConfig.getKeyId(current != null ?
                        current.enterpriseConfig : null));
    } catch (NullPointerException e) {
        throw new IllegalStateException("Invalid config details");
    }
}
```

Graph:

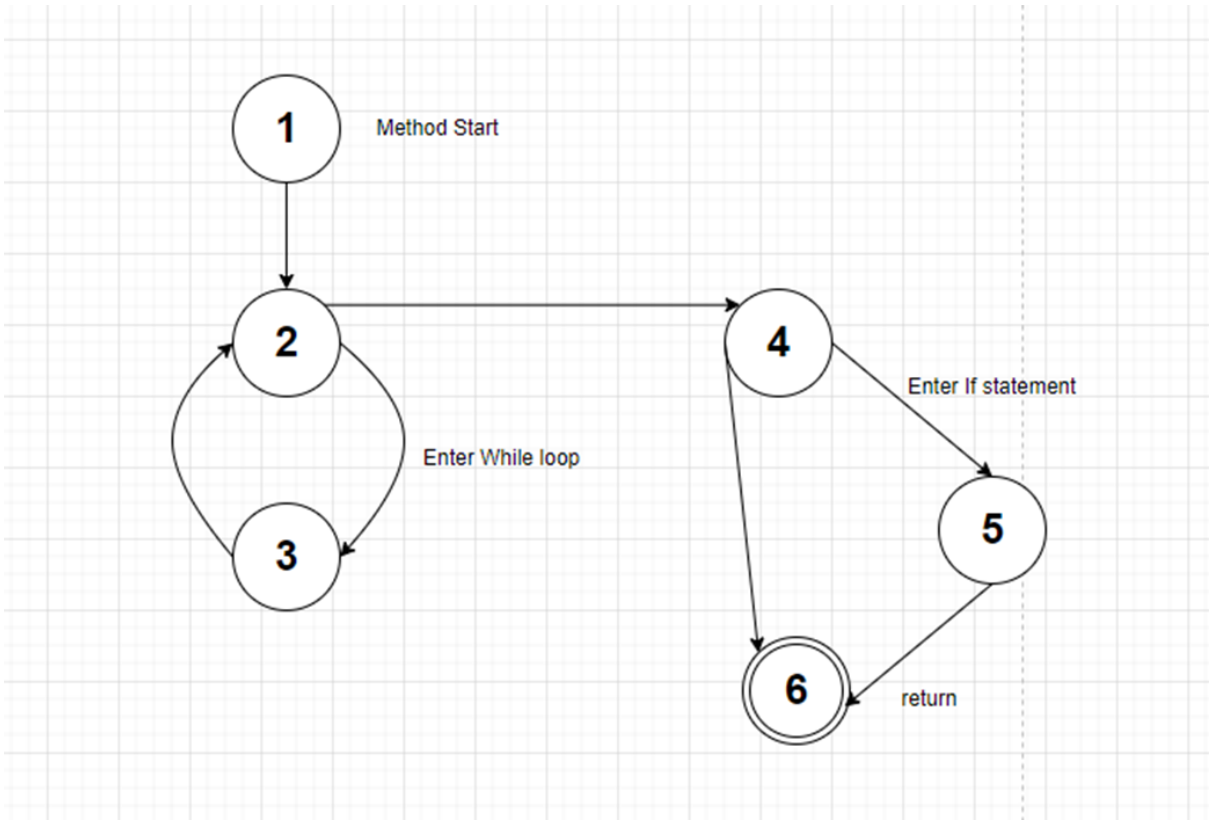

1
TextUtils.isEmpty(SSID)
== true
TextUtils.isEmpty(SSID)
== false
current.SSID throws
NullPointerException
4
2
3
allowedKeyManagement.cardinality()
throws NullPointerException
allowedKeyManagement.cardinality()
== 0
allowedKeyManagement.cardinality()
!= 0
current.allowedKeyManagement throws NullPointerException
5
6
allowedKeyManagement.get(KeyMgmt.WPA_EAP) == true
allowedKeyManagement.get(KeyMgmt.WPA_EAP) == false
KeyMgmt.strings[KeyMgmt.WPA_EAP] throws
ArrayIndexOutOfBoundsException
9
7
8
allowedKeyManagement.get(KeyMgmt.OSEN) == true
allowedKeyManagement.get(KeyMgmt.OSEN) == false
KeyMgmt.strings[KeyMgmt.OSEN] throws
ArrayIndexOutOfBoundsException
10
11
allowedKeyManagement.get(KeyMgmt.IEEE8021X) == true
allowedKeyManagement.get(KeyMgmt.IEEE8021X) == false
KeyMgmt.strings[KeyMgmt.IEEE8021X] throws
ArrayIndexOutOfBoundsException
12
13
allowedKeyManagement.get(KeyMgmt.SUITE_B_192) ==
false
allowedKeyManagement.get(KeyMgmt.SUITE_B_192) == true
KeyMgmt.strings[KeyMgmt.SUITE_B_192] throws
ArrayIndexOutOfBoundsException
14
15
TextUtils.isEmpty(keyMgmt) == false
TextUtils.isEmpty(keyMgmt) == true
16
17
return statement String concatenation does NOT
throw NullPointerException
return statement String concatenation throws
NullPointerException
18

getOrCreateRandomizedMacAddress() source code

```java
/**
 * @hide
 * Returns Randomized MAC address to use with the network.
 * If it is not set/valid, creates a new randomized address.
 * If it can't generate a valid mac, returns the default MAC.
 */
public @NonNull MacAddress getOrCreateRandomizedMacAddress() {
    int randomMacGenerationCount = 0;
    while (!isValidMacAddressForRandomization(mRandomizedMacAddress)
            && randomMacGenerationCount < MAXIMUM_RANDOM_MAC_GENERATION_RETRY) {
        mRandomizedMacAddress = MacAddress.createRandomUnicastAddress();
        randomMacGenerationCount++;
    }

    if (!isValidMacAddressForRandomization(mRandomizedMacAddress)) {
        mRandomizedMacAddress = MacAddress.fromString(WifiInfo.DEFAULT_MAC_ADDRESS);
    }
    return mRandomizedMacAddress;
}
```

Graph of method



Prime Paths

1,2,3

2,3,2

3,2,3

1,2,4,6

1,2,4,5,6

3,2,4,6

3,2,4,5,6

Prime path coverage

1,2,4,6

1,2,4,5,6

1,2,3,2,4,6

Base Choice Coverage

Characteristics

1. mRandomizedMacAddress is null
    a. True
    b. False
2. mRandomizedMacAddress is valid
    a. True
    b. False
3. mRandomizedMacAddress is invalid and not null
    a. True
    b. False
4. MAXIMUM_RANDOM_MAC_GENERATION_RETRY < 0
    a. True

b. False

5.