

UNIVERSITY OF TARTU  
Faculty of Science and Technology  
Institute of Computer Science  
Cybersecurity Masters' Curriculum

Brice Seiler

**Vulnerability of Wi-Fi-enabled Devices to KRACK  
Attacks – A Case Study**

**Master's Thesis (21 ECTS)**

Supervisor: Danielle Morgan, MSc.

Tartu 2023

# **Vulnerability of Wi-Fi-enabled Devices to KRACK Attacks – A Case Study**

## **Abstract:**

In 2017, security researchers Mathy Vanhoef and Frank Piessens discovered a serious set of vulnerabilities in the Wi-Fi Protected Access/Wi-Fi Protected Access 2 (WPA/WPA2) security protocol that became known as key reinstallation attack, also known as KRACK. This set of vulnerabilities allowed attackers to replay, decrypt or forge data transmitted over Wi-Fi. For some Android smartphones, KRACK led to an all-zero key being used, making it trivial for attackers to manipulate the Wi-Fi communication. Although it was acknowledged as one of the most important vulnerabilities against WPA/WPA2, no follow-up studies investigated how devices can be tested against it, and if and how it still affects Wi-Fi-enabled devices today. This Master's thesis conducted a comprehensive analysis of the KRACK vulnerabilities, by investigating their mechanics and detailing how to setup a testing environment to research it. This testing environment was used to examine if devices were vulnerable to one of Vanhoef's seven tests. The seven tests were performed on 29 Wi-Fi-enabled devices collected through a convenience sampling method. In total, 203 test results were gathered. Out of 29 devices, only 2 older smartphones were identified to be vulnerable to the KRACK attack. Network captures are provided for discussing the tests' outcomes.

## **Keywords:**

Key Reinstallation Attack, KRACK, 802.11, WPA/WPA2, 4-way handshake, MC-MITM

## **CERCS:**

T120 - Systems engineering, computer technology

# **Wi-Fi võimekusega seadmete haavatavus KRACK rünnakutele - juhtumi-uuring**

## **Lühikokkuvõte:**

2017. aastal avastasid küberturbe spetsialistid Mathy Vanhoef ja Frank Piessens tõsise haavatavuse Wi-Fi Protected Access/Wi-Fi Protected Access 2 (WPA/WPA2) turvaprotokollis, mida edaspidi nimetati võtme taaspalgaldus rünnakuks ehk “key reinstallation attack”, või KRACK rünnakuks. Antud kogum haavatavusi võimaldas ründajatel taasesitada, dekrüpteerida või võltsida Wi-Fi kaudu edastatud andmeid. Mõne Androidi nutitelefoni puhul viis KRACK selleni, et kasutati kõik nullid võtit (“all-zero key”), muutes ründajate jaoks Wi-Fi kommunikatsiooniga manipuleerimise triviaalseks. Vaatamata sellele, et seda peeti olulisemaks haavatavuseks WPA/WPA2 protokollis, ei ole senini järeluuringuid, kuidas seadmeid antud rünnaku vastu testida või kuidas ja kas see mõjutab veel tänaseid Wi-Fi võimekusega seadmeid. Käesolev magistri töö viis läbi põhjaliku analüüsi KRACK haavatavuste kohta, uurides nende toimemehhanismi ja selgitades, kuidas luua selle uurimiseks testimiskeskond. Antud testimiskeskonda kasutati, et tuvastada seadmeid, mis võivad olla haavatavad ühele Vanhofi seitsmest testist. Antud teste viidi läbi 29-l Wi-Fi võimekusega seadmel, mis koguti testimiseks kasutades mugavusvalimi meetodit. Kokku saadi töös 203 testi tulemust 29-lt seadmelt. Ainult kahel vanemal nutitelefoni tuvastati haavatavus KRACK rünnakule. Töö lõppu on lisatud võrgu pakettide väljavõtted, mis on toeks uurimuse tulemuste üle arutluses.

## **Võtmesõnad:**

Key Reinstallation Attack, KRACK, 802.11, WPA/WPA2, 4-way handshake, MC-MITM

## **CERCS:**

T120 - Süsteemitehnoloogia, arvutitehnoloogia

# Table of Contents

|          |                                                            |           |
|----------|------------------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction .....</b>                                  | <b>5</b>  |
| <b>2</b> | <b>Terms and Notions .....</b>                             | <b>7</b>  |
| <b>3</b> | <b>Background.....</b>                                     | <b>9</b>  |
| 3.1      | Introduction to Wi-Fi Security .....                       | 9         |
| 3.2      | The 4-Way Handshake .....                                  | 11        |
| 3.3      | The KRACK Vulnerability.....                               | 20        |
| <b>4</b> | <b>Literature Review.....</b>                              | <b>29</b> |
| 4.1      | Wi-Fi Security History .....                               | 29        |
| 4.2      | Practical Attacks Against Wi-Fi .....                      | 33        |
| 4.3      | Assessing Wi-Fi Security through Wardriving .....          | 35        |
| <b>5</b> | <b>Problem Statement.....</b>                              | <b>38</b> |
| <b>6</b> | <b>Methods .....</b>                                       | <b>40</b> |
| <b>7</b> | <b>Results .....</b>                                       | <b>44</b> |
| 7.1      | Setup Procedure .....                                      | 44        |
| 7.2      | Script Analysis.....                                       | 47        |
| 7.3      | Devices tested for KRACK Vulnerability .....               | 51        |
| <b>8</b> | <b>Discussion.....</b>                                     | <b>57</b> |
| 8.1      | General Observations .....                                 | 57        |
| 8.2      | Case Studies of KRACK Tests.....                           | 60        |
| 8.2.1    | KRACK against Fairphone FP4 (Android v12).....             | 60        |
| 8.2.2    | KRACK against Blackberry Bold 9700 (BlackberryOS v6) ..... | 62        |
| 8.2.3    | KRACK against Samsung S5 (Android v6).....                 | 66        |
| 8.2.3    | Replaying Broadcast Frames on Surface 3 .....              | 68        |
| 8.3      | Impact and Limits of Results.....                          | 69        |
| <b>9</b> | <b>Conclusion.....</b>                                     | <b>72</b> |

**References ..... I**

**Appendix ..... VII**

    I. Filters used for Wireshark Capture Figures ..... VII

    II. Step-by-step Process of KRACK Tests .....XI

    III. License..... XII

# 1 Introduction

Wi-Fi is one of the most common ways for people to connect to the Internet nowadays, especially for mobile devices: its traffic is estimated to be 5.4 times bigger than that of mobile networks [1]. Defined as a standard in 1997 [2] by the Institute of Electrical and Electronics Engineers (IEEE), Wi-Fi's current economic value is estimated at a staggering 3.3 trillion US\$ [3]. In 2025, it is expected to rise to 4.9 trillion US\$ according to the Wi-Fi Alliance [3]. Some of the identified key contributors are free Internet access through open Wi-Fi networks, consumers accessing the Internet through home routers, and enterprises digitalising their business functions through Wi-Fi [3]. Especially in recent years notably since the 2020 Coronavirus pandemic, Wi-Fi has been heavily used as more and more employees were working remotely [1]. Its main advantage for users resides in its wireless connection, based on radio waves in the 2.4GHz, 5GHz (and now even 6GHz spectra), and its higher data transfer rates and range than other wireless technologies such as Bluetooth [4].

To protect the confidentiality and integrity of Wi-Fi communications, security protocols have been used since 1999 [2]. Even though Wi-Fi security continuously improved over time, a particular set of attacks called “KRACK” (for “Key reinstallation Attacks”) was discovered by researchers in 2017 [5] and was particularly devastating for Wi-Fi security. Indeed, these attacks target a core element of the widely used Wireless Protected Access (WPA/WPA2) security protocols: the 4-way handshake. This process establishes the encryption keys used to secure a connection between a user and a Wi-Fi access point. The 4-way handshake was among the novelties embedded in the WPA and WPA2 security protocols. Soon after KRACK was discovered, the WPA3 security protocol was published.

The authors of KRACK showed that the attack allows malicious actors to replay, decrypt and sometimes even forge Wi-Fi frames, without even knowing the Wi-Fi password of the network. The worst case affects Linux-based operating systems (OSs), such as Android for smartphones: when performing KRACK, an all-zero encryption key was installed and used for communications. This completely breaks WPA/WPA2's security, as the attackers can forge and decrypt any exchanged frames. As shown in [6], Android made up around 70% of the worldwide mobile OS market share in 2017. As of early June 2023, WPA2 is estimated to be the most widely used Wi-Fi security protocol overall. Indeed, it would account for almost 75% of Wi-Fi networks [7] according to wiglet.net (see [Figure 1](#)), one of the most complete cooperative projects for identifying and mapping existing wireless networks in the

world (including Wi-Fi, cell towers and Bluetooth). Comparing some countries, WPA2 represents 72% of the United States' Wi-Fi networks, 78% of Germany's, 80% of Estonia's or 88% of Switzerland's. It should be noted that these statistics might not be representative, as they depend on voluntary contributors scanning their environments for Wi-Fi networks.

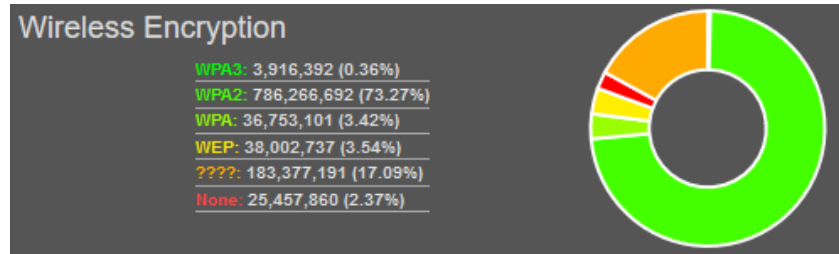


Figure 1: Worldwide Wi-Fi security protocol statistics (as of 04.06.2023) [7]

Taking both the mobile OSs and the supported Wi-Fi security protocols into account, it can be deduced that the potential attack surface for KRACK is important, but little research showcased how to perform this niche attack against Wi-Fi. The findings were perceived as critical [5], as major media outlets started alerting on how broken WPA2 became, although this conclusion was exaggerated. The main objective of this thesis will be to present a comprehensive focus on the KRACK vulnerability and investigate the following: what is it, how does it work, how can it be tested, and is it still affecting devices “in the wild”?

As a preliminary introduction to Wi-Fi security theory, Chapter 3 ([Background](#)) provides an overview of the relevant concepts, namely the cryptographic systems used in Wi-Fi, the underlying steps of the 4-way handshake, as well as the KRACK vulnerability and the related tactic to abuse it, known as Multi-Channel Man-in-the-Middle. Following this theoretical chapter, a literature review on Wi-Fi security and KRACK-related scientific articles is presented. Chapter 4 ([Literature Review](#)) summarises the scientific gap, while Chapter 5 defines the problem statement. This is followed by a description of the methods used to answer the research questions (Chapter 6). Furthermore, Chapter 7 ([Results](#)) presents the details of this thesis' analyses, followed by a discussion of the findings (Chapter 8). Finally, a conclusion summarises this document (Chapter 9). Since many acronyms will have to be used for conciseness, a list of the important abbreviations can be consulted in Chapter 2, following this introduction.

## 2 Terms and Notions

|              |                                                            |
|--------------|------------------------------------------------------------|
| AES          | Advanced Encryption Standard                               |
| AP           | Access Point                                               |
| ARP          | Address Resolution Protocol                                |
| CBC-MAC      | Cipher Block Chaining - Message Authentication Code        |
| CCMP         | Counter mode CBC-MAC Protocol                              |
| CVE          | Common Vulnerabilities and Exposures                       |
| CVSS         | Common Vulnerability Scoring System                        |
| EAPOL        | Extensible Authentication Protocol over Local Area Network |
| GCMP         | Galois-Counter Mode Protocol                               |
| GMK          | Group Master Key                                           |
| GTK          | Group Transient Key                                        |
| HMAC         | Hash-based Message Authentication Code                     |
| IEEE         | Institute of Electrical and Electronics Engineers          |
| ISP          | Internet Service Provider                                  |
| IV           | Initialisation Vector                                      |
| KRACK        | Key Reinstallation Attacks                                 |
| MAC          | Media Access Control                                       |
| MIC          | Message Integrity Check                                    |
| MITM         | Man-In-The-Middle (attack)                                 |
| OS           | Operating System                                           |
| OSI          | Open Systems Interconnection                               |
| PBKDF/PBKDF2 | Password-Based Key Derivation Function (2)                 |
| PMK          | Pairwise Master Key                                        |
| PN           | Packet Number                                              |
| PRF          | Pseudorandom Function                                      |



|          |                                          |
|----------|------------------------------------------|
| PSK      | Pre-Shared Key                           |
| PTK      | Pairwise Transient Key                   |
| RC4      | Rivest Cipher 4                          |
| RSC      | Receive Sequence Counter                 |
| SSID     | Service Set Identifier                   |
| TKIP     | Temporal Key Integrity Protocol          |
| TK/TEK   | Temporal Encryption Key                  |
| WEP      | Wired Equivalent Privacy                 |
| Wi-Fi    | Wireless Fidelity                        |
| WLAN     | Wireless Local Area Network              |
| WNIC     | Wireless Networking Interface Controller |
| WPA/WPA2 | Wi-Fi Protected Access (2)               |

## 3 Background

### 3.1 Introduction to Wi-Fi Security

The 4-way handshake was defined in 2004 in the 802.11i amendment of the IEEE 802.11 standard [8], which introduced WPA and WPA2 to replace WEP. This amendment defines procedures to build so-called Robust Security Networks. WPA was thought of as a temporary solution for compatibility reasons with legacy hardware: it is based on the Temporal Key Integrity Protocol (TKIP) as its data-confidentiality protocol, which uses Rivest Cipher 4 (RC4) as its core cipher. RC4 was the same cipher that was used in WEP, whose implementation was later shown to be weak. In the 802.11i amendment, WPA2 was designed to last longer, as it would use the newly developed Advanced Encryption Standard (AES) cipher within the CCMP block cipher. CCMP stands for Counter mode Cipher block chaining Message authentication code Protocol: it combines the Counter mode with CBC-MAC as its block cipher, and uses AES as its encryption algorithm. The Counter mode is used for ensuring confidentiality, while the CBC-MAC mode is used for ensuring authentication and integrity of the communication. Another amendment (802.11ad), established in 2012, added GCMP (Galois-Counter Mode Protocol) as an optional block cipher. In 2018, yet another security protocol called WPA3 was defined: its authentication method is slightly different from WPA/WPA2 and is called “Simultaneous Authentication of Equals”. Indeed, WPA3 uses an additional type of handshake called “dragonfly handshake”, which uses elliptic curve cryptography and precedes the 4-way handshake. Many attacks that could be used against WPA/WPA2 networks were handled by WPA3 [2], notably the KRACK vulnerability. Nevertheless, other attack techniques against WPA3 were also found later on by researchers [9].

The Wi-Fi protocol (IEEE standard 802.11) is used in both Layer 1 and Layer 2 of the Open Systems Interconnection (OSI) model [10, p. 21]. Layer 1 (or “Physical Layer”) notably defines the frequencies and modulation types used for the radio waves. In Layer 2 (or “Data Link Layer”), concepts like the Media Access Control (MAC) address or the BSS (Basic Service Set) are defined and used. The MAC address is a 12-digit hexadecimal number assigned to each device connected to a network and is determined by the network interface card (NIC). For Wi-Fi communications, a wireless NIC is used (WNIC). Nowadays, devices like smartphones might also use a temporary (random) MAC address instead of the card’s MAC address when connecting to a Wi-Fi network (i.e., a Wireless Local Area Network or

WLAN), for privacy reasons. The BSS corresponds to “a group of stations that communicate with each other” [10, p. 24]. When a user wants to associate its device with a wireless access point (“AP”) to have network connectivity, the BSS is said to be an “infrastructure BSS”. Typically, to connect to such a WLAN, a user will contact the AP and end up having access to the network or the Internet. The Basic Service Set Identifier (BSSID) is the identifier of an AP and by convention, it is its MAC address [11, p. 52]. In 802.11, it is also possible to chain multiple BSSs together to make an Extended Service Set (ESS), so that a user can connect (or keep staying connected) to the same underlying WLAN, although it associates itself to different APs [10, p. 25]. Concerning Wi-Fi security, it’s in Layer 2 that the Wi-Fi encryption mechanism takes place. As stated before, the Wi-Fi protocol uses radio waves to send frames: each frame can be summarised as having a header and a payload, the latter representing the data that will be encrypted.

In Figure 2, the general structure of an encrypted WPA/WPA2 frame is presented. The Layer 2 header is shown at the top, with the encrypted payload (“Payload Data”) delimited as “ciphertext” below it. The 802.11 standard defines different types of frames: Management frames, Data frames and Control frames. Management frames are typically used for establishing a Wi-Fi connection to an AP, the Data frames to transmit data (as expected) while Control frames are used in conjunction with Data frames for acknowledging that the transmission was correctly performed [12, p. 62]. Only Data frames will be encrypted.

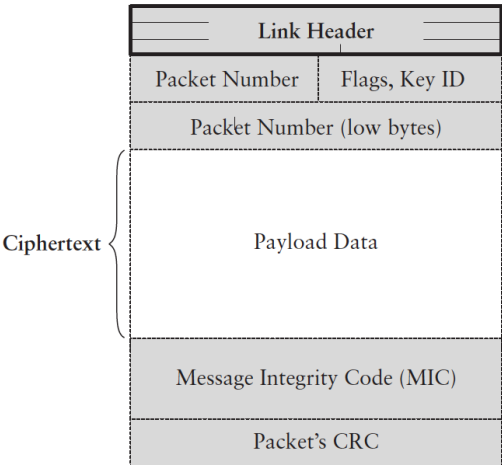


Figure 2: Summarised structure of a WPA/WPA2 frame<sup>1</sup>

<sup>1</sup> Source: <https://securityboulevard.com/2019/10/wpa2-packet-frame-format/> (Accessed 26.04.2023).

The two lines following the “Link Header” part of the frame essentially constitute the data-confidentiality protocol’s header. For each frame, a packet number is defined and used by Wi-Fi encryption schemes (or “data-confidentiality protocols” in the 802.11i standard [13]) to add randomness to the final ciphertext. Note that “packet” and “frame” will be used interchangeably in this thesis. As will be shown in subchapter 3.3, Wi-Fi security protocols implement a unique key for each packet (also called a “per-packet key”), based on the aforementioned packet number. Packet numbers are at the heart of the KRACK attacks and will be discussed thoroughly throughout this thesis.

To get both the client and the AP to use the same encryption key, 802.11i defined an authentication and session key generation process called the 4-way handshake, used in WPA and WPA2. To encrypt data symmetrically, both parties need to exchange information that will prove that they hold the same password, without actually sending it through the communication channel. Using this password, the 4-way handshake will make both parties compute the same session key, which will be used to encrypt communications. This is to protect against attackers who would have cracked previously used session keys, but not the password. The procedure to agree on this session key is described next.

### **3.2 The 4-Way Handshake**

The 4-way handshake was originally developed in the WPA/WPA2 protocols after critical vulnerabilities were discovered in WEP only a few years after its release. WPA is based on the same cryptographic algorithm (RC4) as WEP but in an improved cryptosystem (TKIP). It was thought to be a good intermediate solution to replace the broken WEP, while still being usable by legacy hardware. WPA2 uses a more modern cryptosystem (CCMP), which uses a stronger encryption scheme than TKIP called AES. By default, WPA uses TKIP, while CCMP is WPA2’s default, although it can optionally handle TKIP as well. It is also relevant to outline that WPA and WPA2 both can be used in two different authentication modes: one uses a password (a pre-shared key, or “PSK”) while the other uses an infrastructure that will add another layer of authentication (called “Enterprise”). For example, WPA-PSK uses a password for authentication, while WPA-Enterprise uses a centralized authentication server (also called RADIUS for “Remote Authentication Dial-In User Service”) to authenticate clients. For a given username, the RADIUS server checks if the provided user password is correct, while in PSK mode, all WLAN clients share the same password. The

PSK mode is commonly used at home, while the Enterprise mode is mostly used in larger networks such as in universities.

As the encryption process is symmetric (i.e., each device must have the same password beforehand), it is important to keep the PSK secret. A session key called Pairwise Transient Key (PTK) will be derived from the PSK. The PTK will be unique to the session of a specific client connecting to a specific AP and will be split into subkeys, one of them being the actual key used for encrypting data. Before being able to encrypt the communications, steps need to be taken by a client. The WPA/WPA2 general connection process of a client to an AP is performed in 4 phases [14] pictured in Figure 3. These are:

1. the Network Discovery phase,
2. the Authentication phase,
3. the Association phase,
4. the 4-way handshake.

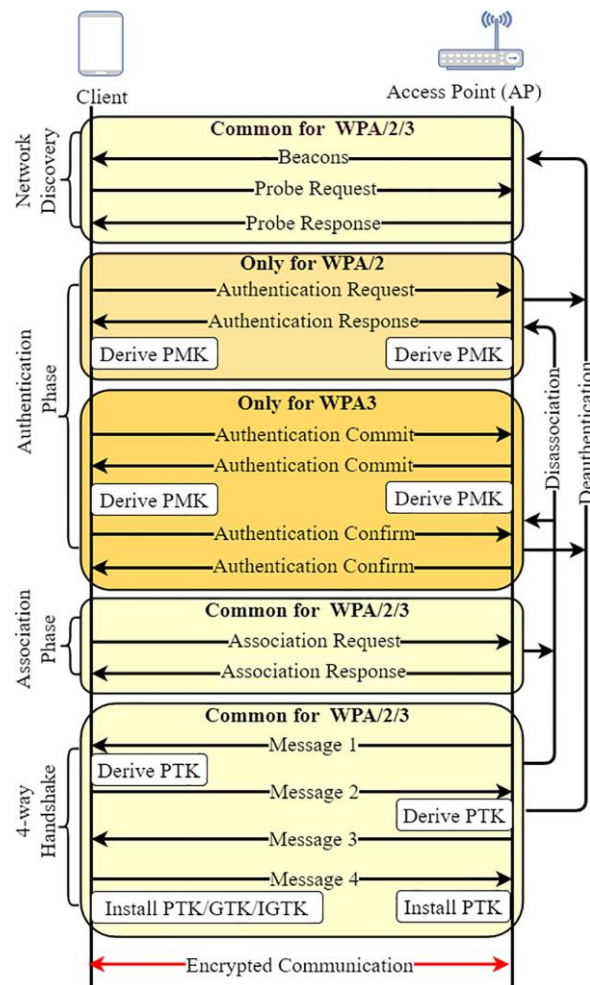


Figure 3: Wi-Fi connection phases [14]

The network discovery (phase 1) consists of APs periodically sending beacons which include among other things, the SSID (Service Set Identifier, i.e., the ESS identifier and more commonly known as the “name” of the Wi-Fi network), the MAC address, and the radio channel it operates on. Clients will scan the channels looking for available APs and provide a list to the user, who will then select the desired Wi-Fi network it wants to use. For that, clients will send a Probe Request frame to the AP to check if the specific network is available, while the AP will answer with a Probe Response frame to acknowledge its availability. In phase 2 and phase 3 (respectively, authentication and association), Open System Authentication is used, which can sometimes be misunderstood: this allows any client to start a connection with an AP, without actually proving its identity (except for its MAC address). It is a relic of WEP. Checking if a user (client) has the right password will indirectly be performed during the 4-way handshake. Following the (open) authentication, association frames will be exchanged telling the other device which cipher suites it will support [14].

Phase 4 is where the security of WPA and WPA2 lies: a mutual authentication is made of 4 messages, thus the name “4-way handshake”. The authentication protocol is based on the IEEE 802.1X standard (also known as Extensible Authentication Protocol or EAP), which implements EAP over LAN (or EAPOL) for Wi-Fi. This is why the 4 messages will also be called EAPOL frames in this thesis. In this standard, the roles are differentiated into a “supplicant” (the client) and an “authenticator” (the AP).

A Wireshark<sup>2</sup> network capture summarises the four phases in Figure 4. It shows the Probe Request/Response exchange between the AP (starting with “Shenzhen”) and the client (with MAC address 22:61:71:23:de:e6). Then, the Authentication phase is taking place, followed by the Association Request/Response frames. Finally, the four messages of the 4-way handshake are exchanged.

| No. | Time        | Source            | Destination              | Protocol | Length | Replay Info                                                           |
|-----|-------------|-------------------|--------------------------|----------|--------|-----------------------------------------------------------------------|
| 139 | 2.968611920 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 802.11 |          | 145    | Probe Request, SN=3051, FN=0, Flags=.....C, SSID="testnetwork"        |
| 140 | 2.973315166 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 802.11 |          | 146    | Probe Response, SN=346, FN=0, Flags=....., BI=100, SSID="testnetwork" |
| 142 | 2.976605207 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 802.11 |          | 70     | Authentication, SN=3052, FN=0, Flags=.....C                           |
| 144 | 2.980321008 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 802.11 |          | 43     | Authentication, SN=347, FN=0, Flags=.....                             |
| 145 | 2.984261141 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 802.11 |          | 163    | Association Request, SN=3053, FN=0, Flags=.....C, SSID="testnetwork"  |
| 150 | 2.993213046 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 802.11 |          | 95     | Association Response, SN=348, FN=0, Flags=.....                       |
| 159 | 3.014150017 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 EAPOL  |          | 146    | 1 Key (Message 1 of 4)                                                |
| 161 | 3.026031524 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 EAPOL  |          | 195    | 1 Key (Message 2 of 4)                                                |
| 163 | 3.029571288 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 EAPOL  |          | 202    | 2 Key (Message 3 of 4)                                                |
| 164 | 3.035435538 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 EAPOL  |          | 173    | 2 Key (Message 4 of 4)                                                |

Figure 4: Wireshark capture of a client’s Wi-Fi connection to an AP

<sup>2</sup> Wireshark is a well-known network analysis tool. More information can be found on this tool at <https://www.wireshark.org/> (Accessed 22.07.2023).

Before going into each message of the 4-way handshake, the hierarchy of keys used in WPA/WPA2 should be understood. Figure 5 summarises it: the password (sometimes called “passphrase”) is used to define the Pre-Shared Key (PSK). Using the PSK, both the AP and the client compute the Pairwise Master Key (PMK) using the PBKDF2 function. PBKDF2 stands for Password-Based Key Derivation Function 2 and needs five inputs: a pseudorandom function (PRF), a password (PSK), a random value (called “salt”), the number of PRF iterations to perform, and finally the desired length of the outputted key (in bytes). In WPA/WPA2, the PRF used is the Hash-based Message Authentication Code (HMAC), the salt value is the SSID, the number of PRF iterations is 4096, and the length of the PMK is 32 bytes, or 256 bits [15].

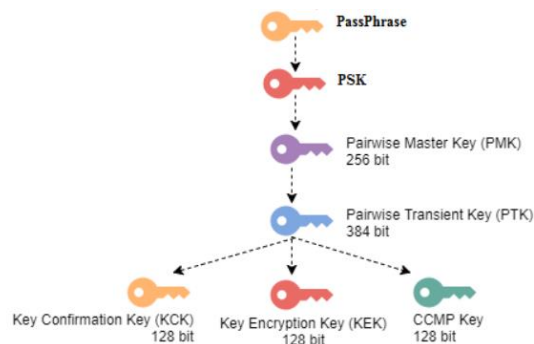


Figure 5: WPA2 Key hierarchy<sup>3</sup>

The session key (PTK) is computed using the PMK and the MAC addresses of the client and the AP, as well as two random values exchanged between the client and the AP. The PTK can only be computed during the session as these two random values are session specific. In brief, using PBKDF2, the PSK gets expanded into a larger random output, which is the PMK. This can be performed before the 4-way handshake starts. Then, using a PRF again, a PTK specific to the connection of a device and its MAC address will be determined.

To be more precise, the PTK is split into different subkeys, the subdivision depending on the Wi-Fi security protocol (WPA or WPA2). In WPA2, the PTK is only split into three subkeys, namely the Key Confirmation Key (KCK), the Key Encryption Key (KEK), and the Temporal Encryption Key (TK or TEK, named “CCMP Key” in Figure 5). In WPA, the

---

<sup>3</sup> Source: <https://praneethwifi.in/2019/11/09/4-way-hand-shake-keys-generation-and-mic-verification/> (Accessed 22.07.2023)

PTK is split into five subkeys: the KCK, KEK, TK, the MIC<sup>4</sup> Tx Key, and the MIC Rx Key. These last two subkeys are specifically used for TKIP in WPA. A summary of the core cryptographic elements used in both WPA and WPA2 is displayed in [Table 1](#).

Table 1 – Keys and Ciphers used in WPA and WPA2

| <i>Parameters per Wi-Fi security protocol</i>                                        | <b>WPA</b>                                                                                                        | <b>WPA2</b>                               |
|--------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| Core Encryption algorithm (default)                                                  | RC4                                                                                                               | AES                                       |
| Cryptosystem (default)                                                               | TKIP                                                                                                              | CCMP                                      |
| PSK      “Pre-shared (master) key”                                                   | WPA/2-PSK mode is used when the client and the AP share a master password (the PSK) before the connection starts. |                                           |
| PMK      “Expanded and randomized version of the PSK”                                | The PMK is computed by expanding the PSK with PBKDF2, with the SSID and the PSK as inputs.                        |                                           |
| PTK      “Session key (unicast)”                                                     | The PTK is the general session key that is agreed upon during the 4-way handshake.                                |                                           |
| PTK subkeys<br><i>Purpose of subkey</i>                                              | Five subkeys make up the PTK (512 bits):                                                                          | Three subkeys make up the PTK (384 bits): |
| 1.    Key to compute EAPOL Message 2’s MIC,                                          | 1. KCK (128 bits),                                                                                                | 1. KCK (128 bits),                        |
| 2.    Key to encrypt GTK in EAPOL Message 3,                                         | 2. KEK (128 bits),                                                                                                | 2. KEK (128 bits),                        |
| 3.    Session key used to encrypt 802.11 frames,                                     | 3. TK (128 bits),                                                                                                 | 3. TK (128 bits).                         |
| 4.    Key to compute TKIP’s MIC for 802.11 frames sent by the AP (only for WPA),     | 4. MIC Tx key (64 bits),                                                                                          |                                           |
| 5.    Key to compute TKIP’s MIC for 802.11 frames sent by the client (only for WPA). | 5. MIC Rx key (64 bits).                                                                                          |                                           |
| GMK      “Master key (multicast/broadcast)”                                          | The GMK is generated by the AP.                                                                                   |                                           |
| GTK      “Session key (multicast/broadcast)”                                         | The GTK is an expanded and randomized version of the GMK.                                                         |                                           |

To encrypt unicast frames, the TK is used. For multicast or broadcast frames, another set of keys is used: the Group Master Key (GMK) is defined by the AP alone and is used during a session to compute a Group Transient Key (GTK). Therefore, the AP will transmit an encrypted version of the GTK for the client during the 4-way handshake for it to use. The

---

<sup>4</sup> MIC stands for “Message Integrity Check” and is computed using either the Michael algorithm in WPA, or a Hash-based Message Authentication Code (HMAC) for WPA2. TKIP also uses MICs in its algorithm [15].



usage of each key will be clarified when presenting the messages exchanged during the 4-way handshake.

In Figure 6, details of the exchanges and computations of the EAPOL messages are displayed. Each message is summarised as “Msg  $n$ ” in Figure 6 ( $n$  representing the EAPOL Message number), with added details on the content of each EAPOL message. Note that it represents the 4-way handshake of a WPA-PSK network, where the PTK is subdivided into subkeys in a different way than for WPA2-PSK as shown before, thus the mention of Tx and Rx MIC keys for the PTK and GTK.

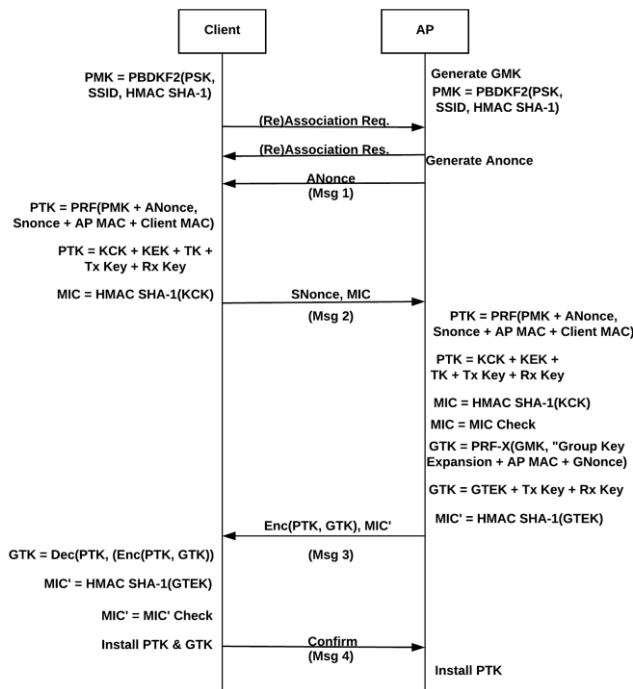


Figure 6: Detailed WPA 4-way handshake and key generation process [2]

Now, let's get into the 4-way handshake process itself, following Figure 6. The 4 EAPOL messages of the handshake are as follows [2]:

1. **EAPOL Message 1** contains the ANonce (“AP Nonce”), a random value generated by the AP<sup>5</sup>. At this point, the client already has all the necessary data to compute the PTK, which will be the session key. Remember that the actual key used to encrypt data will be a subpart of the PTK, namely the TK.

<sup>5</sup> In cryptography, a nonce is a “number-used-once”, meaning a random value, which is often used to add randomness in key-generation processes [2].

2. **EAPOL Message 2** contains the SNonce (“Supplicant Nonce”) and the MIC value computed with a subkey of the PTK, the KCK. Now, the AP is also able to deduce the PTK that will be used for this session. Computing the PTK, the AP will then check if the MIC is valid, and if so, send EAPOL Message 3 to the client.
3. **EAPOL Message 3** contains the Group Transient key (GTK) with another MIC (let’s name it MIC\*). The GTK is the session key used for broadcast and multicast communications inside the local network and is computed with the Group Master key (GMK), pre-generated by the AP. Using the KEK<sup>6</sup> (another subkey of the PTK), the client will decrypt the message and have the GTK and the computed MIC\* value. The client will have to check if the MIC\* matches the HMAC value (i.e., the hash) of the GTK. If that’s the case, it will send Message 4 to the AP.
4. **EAPOL Message 4** contains an “acknowledgement” to confirm that it will use the computed PTK and GTK for further communications. This ends the 4-way handshake. Further Data frames should now be encrypted.

Given the four messages, the PMK is static until the SSID or the PSK is modified, while the PTK (TK) and GTK are static for a given session (i.e., a particular client connected to a specific SSID).

If the wrong password was entered by the user, an incorrect EAPOL Message 2 will be received by the AP, which will send again EAPOL Message 1. A simplified view of the structure of an EAPOL frame is shown in [Figure 7](#). Each EAPOL message (frame) contains a replay counter, which isn’t the same as the packet number, as it is specific to EAPOL frames. It will also play a role in the KRACK vulnerability, which will be explained in [subchapter 3.3](#).

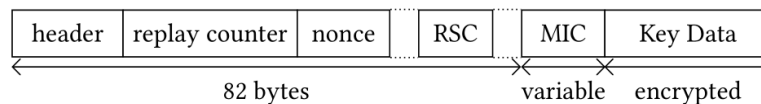


Figure 7: Simplified structure of an EAPOL frame [5]

The AP will increment the replay counter after transmitting an EAPOL frame, and the client will use the same replay counter it received for its response [5]. The EAPOL “nonce” value represents the random nonce the AP (ANonce) and client (SNonce) picked for the 4-way

---

<sup>6</sup> Note that on [Figure 6](#), “PTK” is displayed for encrypting the GTK, while it is in fact the KEK.

handshake, if present in the EAPOL message. Terminology can be confusing, as after the handshake has been completed, the packet number used in each frame by CCMP to protect against replay attacks is sometimes also called a “nonce”. In this thesis, “nonce” will refer to the data-confidentiality protocol’s packet number. The Receive Sequence Counter (RSC) value, sent in EAPOL Message 3 by the AP, is the starting packet number related to the transmitted GTK, used for encrypting multicast/broadcast communications. Any transmitted multicast/broadcast frame during the connection session will have to increment the RSC value. The RSC value is always determined by the AP. If a broadcast/multicast frame contains a packet number lower than the current RSC, then it shall be dropped by the receiver. This will be relevant to a variant of KRACK targeting multicast/broadcast communications. Furthermore, “MIC” represents the MIC value (if transmitted, as in the EAPOL Messages 2 and 3) used to verify the authenticity of the message. Finally, “Key Data” stands for the EAPOL frame’s transmitted key (if a group key is transmitted, like in EAPOL Message 3), which would be encrypted using the KEK. Figure 8 displays the ANonce, SNonce, RSC and related MIC values, as well as the replay counter of the 4 EAPOL messages.

| Source            | Destination       | Protocol | Replay Counter | WPA Key Nonce                                                    | WPA Key RSC      | WPA Key MIC                      | Info                 |
|-------------------|-------------------|----------|----------------|------------------------------------------------------------------|------------------|----------------------------------|----------------------|
| Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 1              | 491b88d9ebca353fa9e99587cbf6c9a349381c1e2a75d8554c3dfd96b6aef33  | 0000000000000000 | 00000000000000000000000000000000 | Key (Message 1 of 4) |
| 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 1              | 0bd756a015801e1d67e4f851a8e54420d3f6f543a1961ef70d5df6c064014e03 | 0000000000000000 | 74824415a3bec50f7a24c254dba73775 | Key (Message 2 of 4) |
| Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 2              | 491b88d9ebca353fa9e99587cbf6c9a349381c1e2a75d8554c3dfd96b6aef33  | 0000000000000000 | 2b70797b7e2b420cbd4a7665b9b1840b | Key (Message 3 of 4) |
| 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 2              | 0000000000000000000000000000000000000000000000000000000000000000 | 0000000000000000 | 9751b111f4c0269d553a26e3d821ba31 | Key (Message 4 of 4) |

Figure 8: The 4 EAPOL frames and relevant data for the 4-way handshake

After the 4-way handshake is performed, all 802.11 Data frames exchanged between the client and the AP will be encrypted. The general encryption scheme is shown in Figure 9. It shows how encrypted frames are constructed after the 4-way handshake was performed.

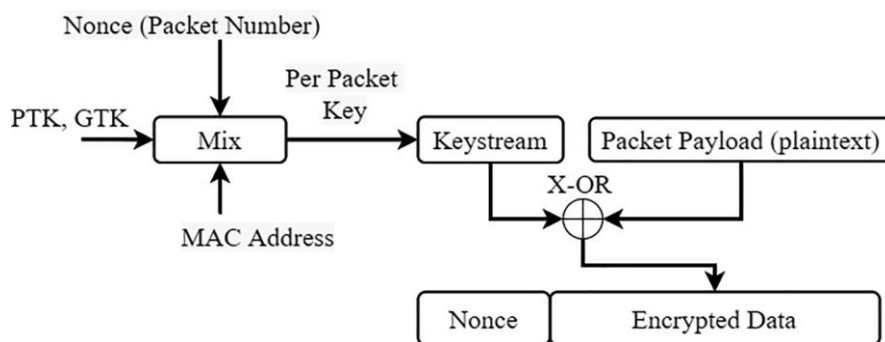


Figure 9: Simplified encryption scheme in WPA/WPA2 [14]

Note that the packet number (PN) is related to the key used: the PTK and GTK will have different ones. Also, the PTK is not the key that will be used to encrypt payloads, but the TK as seen previously. To summarise the process, a binary “exclusive OR” (“XOR”) logical

operation is performed between a (per-packet) keystream and the frame's payload. This means that even if the same payload was encrypted twice, the encryption process would use two different keystreams since the packet number will be incremented for every frame [5]. This is true only if packet number is not reused. The packet number, concatenated to the sender's MAC address and additional flags, which are available in the frame's header, will be used as the Initialisation Vector (IV) of the cryptosystems (CCMP, TKIP). This value acts as a nonce for the security protocols [15]. This should typically ensure that the keystream will be unique to each packet. Since the IV (or nonce) mostly depends on the PN [5], the two will be used interchangeably in this document. The PN gets reinitialised when a new 4-way handshake is established, which should also mean using a new PTK and GTK. The PTK and GTK have their separate PN: the PTK starts with a PN set at 0 (in CCMP) or 1 (in TKIP), while the GTK starts with a PN set at the RSC value given by the AP [5]. For identification reasons, these two types of IVs will be differentiated into "PTK-IVs" and "GTK-IVs" from now on, to make it easier for the reader to follow along.

In brief, the encrypted frames can be summarised as an XOR operation between a per-packet keystream and the packet's payload. The data-confidentiality protocols are considered secure only if the keystream is unique, which is the case if the IV is not repeated for a given key [5]. If the same packet number is reused, then two encrypted payloads will have used the same keystream. Given the XOR operation's definition, if two plaintexts  $P1$  and  $P2$  were encrypted with the same keystream  $C$  into the encrypted payloads  $E1$  and  $E2$ , then it means that it is possible to decrypt one of the plaintexts (e.g.,  $P1$ ) knowing the other plaintext (e.g.,  $P2$ )<sup>7</sup>. As will be shown in the next subchapter, the actual XOR operation can be performed between a known plaintext and its encrypted version to retrieve the keystream. Depending on the data-confidentiality protocol used, IV reuse allows more than that: one can either replay, decrypt or even forge legitimate frames. For this, the session keys need to be reinstalled, which leads to the reinitialisation of the packet number, meaning to a keystream reuse. This trick is at the core of the KRACK vulnerability, presented next.

---

<sup>7</sup> This is well explained in [2]. First, let's define the variables:  $P_x$  represents packet  $x$ ,  $E_x$  represents the encrypted version of packet  $x$ , and that  $C_x$  represents the keystream (or cipher stream) used in conjunction with packet  $x$ . The symbol  $\oplus$  is used as the XOR operator. Now, if  $E_1 = P_1 \oplus C_1$ ,  $E_2 = P_2 \oplus C_2$ , and if  $C_1 = C_2$  then we have that  $E_1 \oplus E_2 = P_1 \oplus P_2 \oplus C_1 \oplus C_2 = P_1 \oplus P_2$ . Since  $E_1$  and  $E_2$  are known to the adversary, it is sufficient for them to know  $P_1$  to find  $P_2$  (compute  $P_1 \oplus E_1 \oplus E_2$ ) or vice-versa (compute  $P_2 \oplus E_1 \oplus E_2$ ).

### 3.3 The KRACK Vulnerability

Now that the different keys and their usage should be more familiar to the reader, the concept of the Key Reinstallation Attack (KRACK) vulnerability is presented. It encompasses a list of 10 Common Vulnerabilities and Exposures (CVE) codes. CVE codes are vulnerability identifiers and are widely used to assign criticality scores based on various factors. The scale goes from 1 (“Low”) to 10 (“Critical”). As a pre-emptive note, the concepts related to CVE-2017-13081 (IGTK), CVE-2017-13084 (PeerKey handshake), CVE-2017-13086 (TDLS), and CVE-2017-13087/8 (WNM) won’t be discussed further in this thesis.

The full list is as follows [16]:

- CVE-2017-13077: Reinstallation of the pairwise encryption key (PTK-TK) in the 4-way handshake.
- CVE-2017-13078: Reinstallation of the group key (GTK) in the 4-way handshake.
- CVE-2017-13079: Reinstallation of the integrity group key (IGTK) in the 4-way handshake.
- CVE-2017-13080: Reinstallation of the group key (GTK) in the group key handshake.
- CVE-2017-13081: Reinstallation of the integrity group key (IGTK) in the group key handshake.
- CVE-2017-13082: Accepting a retransmitted Fast BSS Transition (FT) Reassociation Request and reinstalling the pairwise encryption key (PTK-TK) while processing it.
- CVE-2017-13084: Reinstallation of the STK key in the PeerKey handshake.
- CVE-2017-13086: Reinstallation of the Tunnelled Direct-Link Setup (TDLS) PeerKey (TPK) key in the TDLS handshake.
- CVE-2017-13087: Reinstallation of the group key (GTK) when processing a Wireless Network Management (WNM) Sleep Mode Response frame.
- CVE-2017-13088: Reinstallation of the integrity group key (IGTK) when processing a Wireless Network Management (WNM) Sleep Mode Response frame.

The attacks against the group key handshake (CVE-2017-13080) and FT handshake (CVE-2017-13082) will be briefly presented further in this subchapter, after having explained the concept behind the KRACK attacks. Also, the group key handshake will be summarised. As can be seen in the list, all these vulnerabilities consist of a type of key reinstallation. The main CVE is considered, according to Vanhoef, to be CVE-2017-13077 as it is the most critical one in terms of potential harm against a Wi-Fi user. Based on the Common Vulnerability Scoring System version 3 (CVSSv3), it received a score of 6.8 (“Medium”) from the National Institute of Standards and Technology (NIST), a US governmental agency,

following the article of Vanhoef and Piessens [5]. Before their paper, the most efficient way to attack WPA and WPA2 networks directly was to run dictionary attacks<sup>8</sup> against them, meaning that the Wi-Fi password (PSK) could be easily guessed or found using lists of possible passphrases to test against the AP [2]. More details on the evolution of Wi-Fi attacks are presented in the literature review (Chapter 4). This type of attack usually needs to capture the 4-way handshake between the client and the AP to work, as the encrypted communication that will follow uses the TK (a subkey of the PTK representing the “session” key), and not the PSK nor the PMK. Plain brute forcing is hardly feasible with current technologies and strong passwords [2]. Capturing the 4-way handshake was the main type of attack against WPA/WPA2 (also called “offline dictionary attacks”) until the various key reinstallation attacks were observed under certain conditions. With KRACK, an attacker doesn’t need to capture a 4-way handshake or retrieve the PSK but only needs to be in the range of both the client and AP. It uses an adversarial technique called a Multi-Channel Man-In-The-Middle (MC-MITM) attack. The core elements behind KRACK are explained first, followed by a brief explanation of the MC-MITM attack.

The core vulnerability resides in the machine state of devices. The supplicant goes through various states depending on conditions during the 4-way handshake (Figure 10).

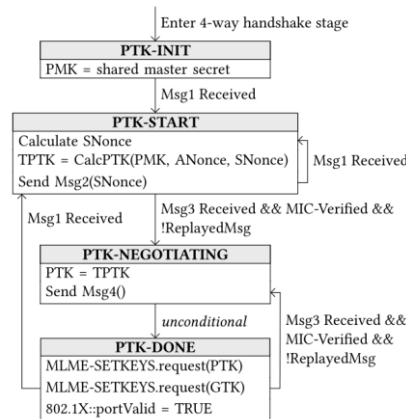


Figure 10: Informal supplicant (client) machine state during the 4-way handshake [5]

As explained in Vanhoef and Piessens’ article [5], this state machine of the supplicant (i.e., the client) concerning the 4-way handshake was specified in the 802.11r amendment in

<sup>8</sup> A dictionary attack is a technique that aims at improving brute-force attacks by giving a list of possible wordlists to test (and possibly, combinations of these words by defining rules), instead of going over all possible values.

2016. First, the supplicant enters the “PTK-INIT” state when starting the 4-way handshake, after the Authentication and Association steps were performed. The PMK is computed as detailed in the previous subchapter. Then, when EAPOL Message 1 is received by the supplicant, the PTK’s computation is performed. The supplicant will move to the “PTK-START” state. In [Figure 10](#), “TPTK” stands for “Temporary PTK” and is used because the supplicant still needs confirmation from the authenticator that it will be the valid PTK. The supplicant’s last action in this state is to send EAPOL Message 2 to the authenticator. If this message is not received by the authenticator, it will resend EAPOL Message 1 to the supplicant. If EAPOL Message 1 is received again, the supplicant will stay in the same state.

Following this, the supplicant will switch to the “PTK-NEGOTIATING” state if the following conditions were satisfied: it received EAPOL Message 3, the authenticator’s MIC was successfully verified, and the EAPOL replay counter is validated. In this state, the supplicant will confirm its PTK (switching from a TPTK to PTK) and send EAPOL Message 4 to the authenticator. Automatically, the supplicant will then move to the PTK-DONE state where it will install the session keys, namely the PTK and GTK. It also enables the 802.1X logical port for the supplicant, used for sending and receiving encrypted frames. If EAPOL Message 3 is sent again (and it is valid, meaning it has a correct MIC and a valid replay counter value), then the supplicant goes back to the “PTK-NEGOTIATING” state, thus resetting the PTK to the TPTK. On the authenticator’s side, the 802.11i standard states that to accept EAPOL Message 4 (i.e., the EAPOL replay counter was validated), the AP should check that “[the replay counter] *was one used in the handshake*” [5]. Finally, the retransmission of EAPOL Messages 1 and 3 by the AP can also happen if it respectively didn’t receive EAPOL Messages 2 and 4. Note that this can happen also because of background noise.

The vulnerability resides in the fact that the supplicant can still accept EAPOL Message 3 being retransmitted to it even when it is in the last state (“PTK-DONE”), with the PTK and GTK already installed using the MLME.SETKEYS primitive function. This will lead to the following scenario: after the client sends EAPOL Message 4, it will automatically install the agreed PTK. It also means that it will normally<sup>9</sup> start encrypting any Wi-Fi frame it will

---

<sup>9</sup> The wording “normally” is selected here on purpose because, as shown by Vanhoef and Piessens [5], the exact implementation of 802.11i is not uniform across operating systems, supplicants, and APs. The paper also discusses the case where the retransmission of EAPOL Message 3 to the supplicant will be discarded if the retransmitted EAPOL Message 3 is not encrypted using the previously agreed PTK. For the same reasons, it

send to the AP using its PTK, most of the time also including the first transmission of EAPOL Message 4 according to [5]. To perform the attack, the adversary must block the arrival of EAPOL Message 4 from the client to the AP, so that the latter resends EAPOL Message 3 to the client, as per the 802.11 standard. The attacker will transfer this message to the client. When this happens, the client will reinstall the (same) session keys, with the nonce (packet number) used by the data-confidentiality protocol (TKIP, CCMP, GCMP) also being reset. As a result, the client will send EAPOL Message 4 to the AP again, but this time encrypted using the previously agreed PTK. This leads to an example of keystream reuse that an adversary can use against the cryptosystem, as shown in Figure 11.

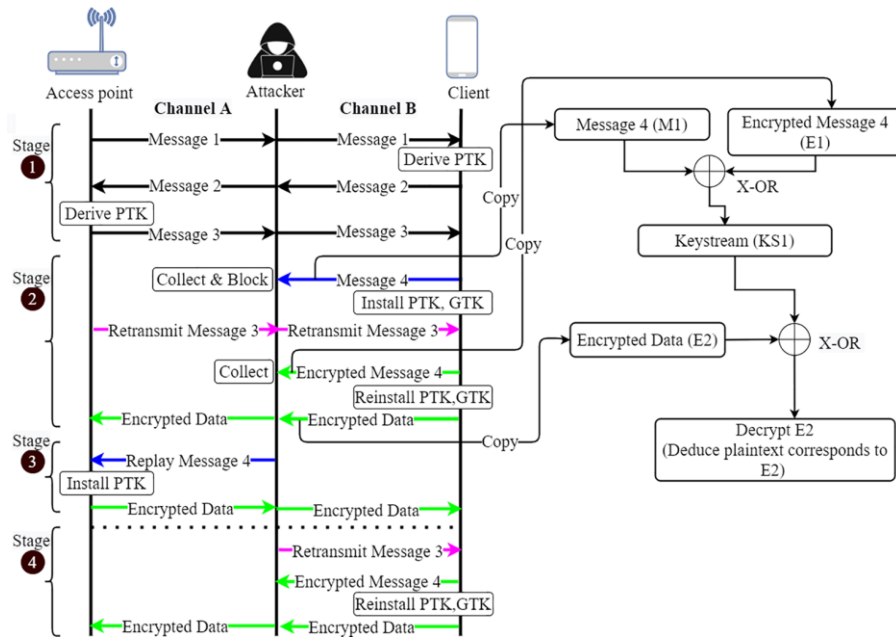


Figure 11: Example of attack using the KRACK vulnerability with a MC-MITM position [14]

Indeed, by accepting the retransmission of EAPOL Message 3 (the pink arrows in Figure 11), the client will reset the packet number. In Figure 11, keystream reuse is exemplified when EAPOL Message 4 is first transmitted unencrypted at the end of the handshake, with its encrypted version being transmitted by the client after having received EAPOL Message 3 again. Therefore, an attacker can use the invertible XOR operation on the encrypted payload with the plaintext payload (here, EAPOL Message 4) to retrieve the keystream for this packet number. More generally, Vanhoef found that this keystream reuse allowed him to

---

is also possible that the AP rejects the encrypted EAPOL Message 4 sent by the supplicant, since it didn't install its keys yet (i.e., because the plaintext EAPOL Message 4 wasn't received yet).



either replay, decrypt or forge packets depending on the data-confidentiality protocol (see Figure 12).

|                     | Replay <sup>c</sup>      | Decrypt <sup>a</sup> | Forge                    |
|---------------------|--------------------------|----------------------|--------------------------|
| <i>4-way impact</i> |                          |                      |                          |
| TKIP                | AP → client              | client → AP          | client → AP <sup>b</sup> |
| CCMP                | AP → client              | client → AP          |                          |
| GCMP                | AP → client              | client → AP          | client ↔ AP <sup>b</sup> |
| <i>FT impact</i>    |                          |                      |                          |
| TKIP                | client → AP              | AP → client          | AP → client              |
| CCMP                | client → AP              | AP → client          |                          |
| GCMP                | client → AP              | AP → client          | AP ↔ client <sup>b</sup> |
| <i>Group impact</i> |                          |                      |                          |
| any                 | AP → client <sup>c</sup> |                      |                          |

<sup>a</sup> With this ability, we can hijack TCP connections to/from an Internet endpoint and inject data into them.

<sup>b</sup> With this ability, we can use the AP as a gateway to inject packets towards *any* device connected to the network.

<sup>c</sup> This denotes in which direction we can replay unicast and group-addressed frames. For the group key handshake, only group-addressed frames can be replayed.

Figure 12: KRACK impact by data-confidentiality protocol used [5]

As was shown in the list of CVEs, one targeted the FT handshake (CVE-2017-13082). As Vanhoef developed a test for the FT handshake vulnerability, it is worth briefly summarising it now. The FTK handshake also consists of four steps (stage “1” in Figure 13).

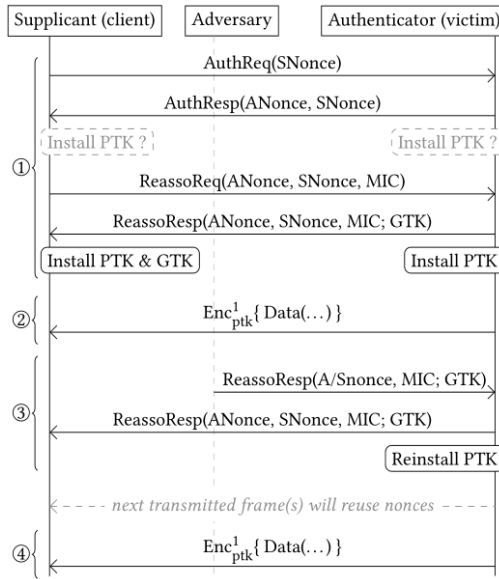


Figure 13: KRACK<sup>10</sup> against the FT handshake [5]

To allow for a faster transition between APs of the same Wi-Fi network (i.e., part of the same ESS), the 802.11r amendment was developed and defined the FTK handshake. The

<sup>10</sup> In stage “3”, the first frame should be ReassoReq and not ReassoRep (see Vanhoef’s Erratum in [5]).

first two messages are an Authentication Request (AuthReq), and an Authentication Response (AuthResp), which carry randomly generated nonces used to derive a fresh PTK. Following this, the client sends a Reassociation Request (ReassoReq), and the AP replies with a Reassociation Response (ReassoResp). A MIC is used to authenticate the two reassociation messages. Furthermore, no replay counter is used in the FT handshake. Instead, it relies on the (random) SNonce and ANonce values to provide replay protection between different handshake sessions. The 802.11r amendment also states that the PTK must be installed after the AuthResp message is sent or received. Additionally, the 802.1X logical port is opened only after sending (or receiving) the ReassoReq message. This ensures that the AP and client will only transmit or accept 802.11 data frames after a handshake got completed, even if the PTK was already installed. As such, the FT handshake should in theory be protected against KRACK. According to Vanhoef, this was apparently not the case in reality: the PTK (and GTK) are reinstalled when replaying a ReassoReq message. It can be replayed since there are no replay counters and the MIC is valid. The AP will accept and process this replayed frame, and thus reinstall the keys and reset the related IVs.

The last handshake presented in this subchapter is the group key handshake. CVE-2017-13080 targets this handshake. While the GTK is first installed during the 4-way handshake, it can be periodically refreshed by the AP during the so-called group key handshake [13, p. 220]. It only consists of two EAPOL messages: the EAPOL “group” message 1 is sent by the AP to all clients, while the second “group” message is sent by a client to the AP, acknowledging this change. If the AP didn’t receive a reply to its request, it will retransmit EAPOL “group” message 1. The EAPOL “group” message 1 contains the new GTK to use, and normally the RSC value of the last used GTK-IV. In reality, the AP determines which RSC value to send to clients: the client will accept it if the RSC value “*has not yet been seen before*” [13, p. 221]. In response, the client will send the second group message with an incremented RSC. The idea behind the attack of this handshake is to collect a retransmitted EAPOL “group” message 1, block it from arriving at the client, and forward it to the client at a later stage. This will trick the client into reinstalling the GTK, and thus reinitialise the GTK-IV of the installed group key. When clients want to send broadcast or multicast frames (i.e., group frames), they will essentially send unicast frames to the AP, who will then use the GTK to send actual encrypted broadcast/multicast frames on the WLAN. For clients, the GTK is therefore only used to decrypt AP-encrypted multicast/broadcast frames. Since the key reinstallation targets the client, no nonce reuse can be forced. Instead, the GTK

reinstallation makes it possible for the attacker to replay frames from the AP since the RSC value can still be reset [5]. The impact might be indirect but nonetheless important: as mentioned by Vanhoef in [5], taking the Network Time Protocol<sup>11</sup> operated in broadcast mode as an example, replaying frames could compromise time reliability and result in target devices being stuck at a certain time if the replay is periodical. Vanhoef mentions that unreliable clock time can impact other security systems like TLS certificates, DNSSEC, or Kerberos authentication.

As mentioned previously, the adversary performs an MC-MITM attack to set itself between the client and the AP. As thoroughly explained in [14], this attack allows the previously established and secured connection between a client and the AP to remain untouched by the adversary. Instead, it will use two Wi-Fi channels, thus the name “Multi-Channel”. The attacker will spoof the real AP for the client on another channel while spoofing the client on the initial channel for the AP, as summarised in Figure 14.

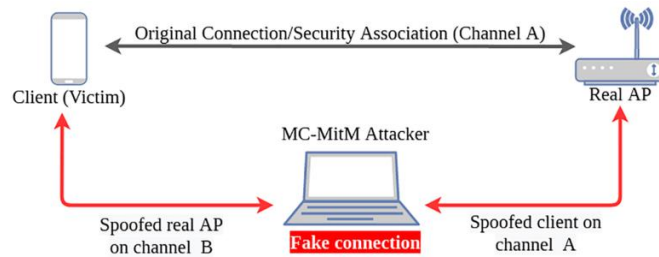


Figure 14: General configuration of an MC-MITM attack [14]

To perform an MC-MITM, an adversary first must set up (rogue) interfaces to spoof the client and the AP, and secondly, force them to switch to the selected rogue channels. To perform the first step, the adversary needs to clone the AP on another channel than the currently used one, meaning that it will set up a rogue AP using, among other things, the same MAC address and the SSID as the legitimate AP. To seem legitimate to the AP, the adversary needs to modify the firmware of its second Wi-Fi interface to send acknowledgement Wi-Fi frames when it receives unicast frames from the AP (sent to the legitimate client), thus impersonating the real client. The result should be that a rogue client (interface 2) listens on channel A for the real AP, while a rogue AP (interface 1) communicates on channel B with the legitimate client. This technique allows an attacker to stay stealthy, as the only

<sup>11</sup> This protocol is used as a networking protocol helping computers’ clock times to be synchronized inside of a network.

visible change is the channel used (for communications with the legitimate client). The two interfaces used by the adversary will just transfer frames from one to the other. The previously secured communication channel remains intact, as the goal is to move the client to the new channel and to impersonate the legitimate client for the AP on the same Wi-Fi channel as before.

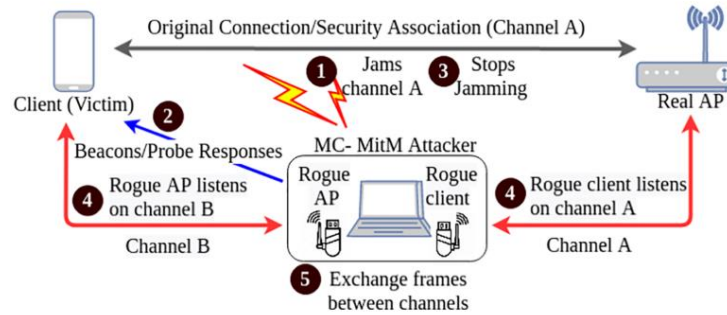


Figure 15: Steps to perform an MC-MITM attack [14]

To get the client to move to the new channel, beacons from the legitimate AP are retransmitted (Figure 15). This can be performed using Channel Switch Announcement frames and work if the attacker's signal is stronger than the AP's. To push the client to the new channel, the attacker can temporarily jam the legitimate channel so that the client sees the beacons of the rogue AP. The client, trying to find the AP by scanning other channels, will find (replayed) Beacon frames from the legitimate AP with the correct SSID on another channel, and will in practice end up sending a Probe Request on this channel, in fact, to the attacker's AP. An easier alternative is to send Channel Switch Announcement frames to the client with a stronger signal than the ongoing communications to and from legitimate AP, which will also move the client to channel B [14].

Following this, the rogue AP sends a custom Probe Response to the client on the new channel B to have the client continue the process with an Authentication Request frame sent to the rogue AP. Then, the rogue AP can collect this Authentication Request frame and retransmit it to the true AP on the legitimate channel, using the rogue client. The real AP should accept it, and in response, will send an Authentication Response frame on channel A. This leads to the rogue client collecting and retransmitting this response on the new channel. Analogously, Association frames are retransmitted through the two interfaces. The 4-way handshake can take place following the completed association step. In summary, the communication channel is only interrupted for a short period of time so that the client starts

communicating on the new channel and sends an authentication request frame, which will be sent to the legitimate AP on the previous channel [14].

## 4 Literature Review

The present chapter aims at giving a literature overview of the available research on Wi-Fi security and KRACK-related research questions. First, the history of Wi-Fi security preceding and surrounding the KRACK discovery is presented, supplemented by articles on the perception of Wi-Fi-related risks. This is meant to present a historical overview of previous research on Wi-Fi security challenges to the reader, as well as the user-perceived risks of using Wi-Fi under certain conditions. Then, a summary of practical attacks against Wi-Fi security mechanisms is suggested. This subchapter showcases articles that explain ways to find new attacks or implement already known attacks, against Wi-Fi networks. Finally, it is followed by a presentation of articles that assessed the nearby Wi-Fi security environment, mainly through wardriving. It seems to be the most popular technique to collect large amounts of data on Wi-Fi-enabled devices and helps to assess the Wi-Fi security environment. As will be shown, some research groups looked at how ISPs (Internet Service Providers) provide (or do not) Wi-Fi security measures to their customers, while others mainly focused on Wi-Fi security statistics.

### 4.1 Wi-Fi Security History

Previous studies already showed ways to get around some of the Wi-Fi security measures put in place, starting with WEP. A few years after its release, WEP's core encryption scheme, RC4, was shown to have various vulnerabilities embedded in it [17]. Exploiting these, researchers developed attacks against the WEP encryption key, which would need the collection of encrypted Wi-Fi frames [18]. This is always possible as Wi-Fi frames are radio waves and thus physically reachable if an attacker is in range. Studies continued to implement more efficient techniques, with the peak of this development being the attacks described by research groups in 2007 [19] and 2008 [20]. They showed that WEP cracking could take less than 1 minute, because of its weak cryptosystem.

This called for a quick replacement, which was presented by the IEEE committee responsible for 802.11. They proposed two new security protocols: WPA, which would still use RC4 but in a different way (thus, remaining compatible with legacy hardware), and WPA2, which would use the more secure AES algorithm for future use in hardware. The fact that WPA's security was still using RC4 made it vulnerable to similar WEP attacks, as showed by the same German research group in [20] in 2008, although they needed the password to be "weak".

While WEP and WPA attacks were already discussed in various papers, another Wi-Fi security certification meant to be easier to set up was soon shown to be also easier to hack: the Wi-Fi Protected Setup (WPS) system, released in 2007. It was developed by Cisco and implemented as an alternative way of connecting clients to a Wi-Fi router that uses WPA/WPA2. Indeed, a PIN code is used to authenticate clients instead of a password. A couple of years after its release, a security expert showed how poorly the WPS system was designed and implemented. He showed that brute-forcing the PIN was possible given its short length, leading to a maximum of 11'000 possible values [21]. The attack wouldn't be affected by the length of passwords, and thus, if no blocking scheme was implemented on the router's software, the attack could take less than 5 seconds. He originally designed a program in Python as a proof-of-concept, described in his article. Soon, tools hacking WPS would be developed and included in the Kali OS as well<sup>12</sup>.

As presented in subchapter 3.2, the WPA and WPA2 protocols are using the so-called 4-way handshake, with the security of the system thoroughly analysed in [22]. Given its key generation scheme, just capturing packets wouldn't suffice to find the encryption key, although it will be later shown that specific attacks could still be run against it. The attention was moved to the 4-way handshake itself, as it included all the needed elements to recover the password by brute-forcing the handshake, which is feasible when the PSK is weak. The most popular technique was thought to be launching a so-called dictionary attack on the captured handshake to uncover the initial password. This is so because WPA and WPA2's schemes to derive the PTK and PMK using the PSK are time-consuming processes, thus making plain brute-force attacks almost unfeasible. A well-known tool package for dictionary attacks against WPA/WPA2 is the *aircrack-ng* suite<sup>13</sup>, but it isn't the only one. Next to WPA-PSK and WPA2-PSK Wi-Fi configurations, there are also the WPA-Enterprise and WPA2-Enterprise networks, which use a per-user password to derive a session key. The literature on attacks targeting these systems was found to be less abundant than for the PSK systems. Nevertheless, studies showed that the Enterprise mode also has its vulnerabilities, as it will be briefly shown later.

---

<sup>12</sup> The main tool to hack WPS on Kali is called *Reaver* and is included by default in the OS. For more information, see <https://www.kali.org/tools/reaver/>.

<sup>13</sup> See <https://www.aircrack-ng.org/> for more details.

As shown previously, the KRACK vulnerability was discovered by Vanhoef and Piessens in 2017 [5]. This attack allows a user to replay, decrypt captured messages and sometimes even forge messages (unless CCMP is used). The authors discovered that most Wi-Fi-enabled devices (using various OSs) were vulnerable to this attack. For Linux-based OSs, typically Android for smartphones, the attack was even worse as the reinstalled key (TK) was always an all-zero-byte key. This is due to a mechanism that was thought to be good for security: after having been installed, the PTK was deleted from memory by setting its corresponding value to 0 bytes to reduce the possibility of malicious access to it. By doing so, when KRACK is performed, the PTK (and thus the TK) is set to all-zero bytes. The vulnerability resides in the *wpa.c* file as explained in [23], referring to the commits that were performed by the authors of the *wpa\_supplicant* and *hostapd* packages<sup>14</sup>. Vanhoef also raised the specific issue of the *wpa\_supplicant* client on the Chromium OS reinstalling all-zero keys in 2017, showcasing how to check for vulnerable devices [24]. This vulnerable behaviour affects *wpa\_supplicant* v2.4, v2.5 and v2.6. Vanhoef also published another study, written in 2018, describing improvements in performing the KRACK attacks, in particular at the implementation level of these attacks [25], but also showing that other types of related handshakes were vulnerable to KRACK.

In 2019, a Master's thesis performed at the Czech Technical University in Prague also thoroughly discussed KRACK and detection mechanisms [26]. Its listed goals were to develop a testing tool, study generated traffic while KRACK is performed, as well as analyse detection mechanisms. Also, testing KRACK against Wi-Fi-enabled devices was performed. The development of a KRACK-exploitation tool was reduced to a detection system, similar to the scripts Vanhoef published later on his GitHub repository [27]. The code was written in Python and is not publicly available. Also, the same Vanhoef scripts were used in this thesis to test KRACK on devices. Out of 21 devices, 4 devices were reinstalling the PTK, all using Android v6 or below. Interestingly, all the devices were apparently reinstalling the GTK during the 4-way handshake. It seems that only the main testing script was tested. The group key handshake wasn't tested for example. The script to test the FT handshake was tested on two university APs. None of them were vulnerable.

---

<sup>14</sup> Actually, the specific commit number `ad00d64e7d8827b3cebd665a0ceb08adabf15e1e`, available at <https://w1.fi/cgit/hostap/commit/?id=ad00d64e7d8827b3cebd665a0ceb08adabf15e1e> (Accessed 11.06.2023), was used for the analysis in [23].



Another important 2022 paper [14] concerns the technique which Vanhoef and Piessens' used for discovering KRACK in [5], namely the MC-MITM. The researchers present a review of the technique, which is at the core of KRACK exploitation, and discuss the various ways it can be used to attack Wi-Fi networks in general. The authors have a subchapter on the KRACK vulnerability. Reviewing types of MC-MITM attacks and patching statistics, the authors cite tests from a cybersecurity company done in 2019, from which they deduced interesting statistics: testing multiple Wi-Fi-enabled devices (wireless networking cards, Wi-Fi client software, AP, OS, etc.), around 90% of them were still vulnerable to KRACK (although the vulnerabilities were 2 years old). Using the same MC-MITM technique, Vanhoef found new vulnerabilities related to packet fragmentation, which he nicknamed "FragAttacks" [28]. Although the conditions for it to work are stricter than for KRACK, he found that it affected all wireless security protocols, from WEP to WPA3.

Another vulnerability similar to KRACK was later published in 2020 by a Slovak research team working at ESET, a company specialising in IT security software [29]. While the attack, named Kr00k, is similar to KRACK, the vulnerability resides at a different level. In fact, the Kr00k vulnerability exists because when disassociating, some wireless network cards (of clients and Wi-Fi routers included) were "flushing" the current encryption key, setting it back to all-zero bytes. At the same time, the device will also empty its memory, sending the remaining data encrypted with the key, which became an all-zero key. This means that some remaining frames will be encrypted using this all-zero key, similar to the Linux-Android vulnerability to KRACK.

To conclude this subchapter with the user perception of Wi-Fi-related risks, a 2020 research paper [30] qualitatively analysed the Wi-Fi security awareness of users, with a specific emphasis on KRACK. Their qualitative method consisted of semi-structured interviews with Wi-Fi router owners in the area of Munich, with a small sample of 16 participants, a majority being male students. The authors found that 37.5% of Android devices (6/16) and 62.5% of iOS devices (5/8) were vulnerable to KRACK, although how this was determined is unknown. Also, they concluded that although most participants were aware of Wi-Fi-related risks, they didn't implement security best practices when using or managing their Wi-Fi infrastructure, such as changing factory default settings or having strong passwords. Another research group pursued similar research objectives in Japan in 2018, by setting up open Wi-Fi networks and looking at the traffic of users that would connect to them [31]. They observed that women, and more generally people with higher education, were more

likely to use public Wi-Fi and that the main motivation for taking this risk was to preserve their mobile data allowance, especially for people with a low monthly data allowance. They also noticed that many user applications still didn't add encryption for their communications, which would indicate that Wi-Fi security is still relevant to this day to ensure confidentiality. It should be pointed out that these results might encompass cultural and regional biases that aren't addressed thoroughly in their discussion.

## **4.2 Practical Attacks Against Wi-Fi**

With time, research studies started to develop specialised attacking tools and techniques. For example, one paper written in 2015 [32] presented a way to attack WPA2-PSK without capturing the handshake. It speeds up the dictionary attack on WPA2-PSK by creating multiple virtual wireless clients (VWC) that would "act" as real clients. Each of the VWCs would then make password guess trials: each VWC will reply to the AP's EAPOL Message 1 with an EAPOL Message 2, and if the password was wrong (i.e., a new EAPOL Message 1 is sent by the AP), then it would try another password, until a Deauthentication frame is sent by the AP. The authors tested their script live on various wireless router models, and only needed one WNIC. They saw that they could speed up the process by a factor of 100. The issue with this methodology is that it was heavily dependent on the routers' processing power, although being an interesting alternative if no 4-way handshake can be captured.

Other such examples are the various side-channel attacks against WPA-TKIP developed by Schepers, Ranganathan, and Vanhoef in 2019 [33]. First, the research looked at the implementation of TKIP at several levels of the Wi-Fi stack, such as the hardware (the WNIC), the OS (i.e., the driver or Wi-Fi subsystem) and the Wi-Fi client. Then, focusing on the attack on TKIP discovered in 2009 by [20], they aimed for ways around the deployed countermeasures and successfully found them on numerous elements of the Wi-Fi stack. Finally, the research group also considered the attack surface of their attacks and compared it between different regions, namely Boston (USA), Hasselt (BE), and Leipzig (DE). To collect the APs' data, they used wardriving, a methodology consisting of driving around in towns in a car with Wi-Fi dongle-connected hardware (usually, compact ones like an Arduino or Raspberry Pi system). The goal of this method is to quickly gather information about surrounding APs, such as the SSID, the channel the Wi-Fi network is using, or the type of security protocols it supports. They found that on average, almost 50% of identified routers were configured with WPA-TKIP, with stark differences between regions (+30% in Hasselt

compared to Leipzig). Their side-channel methods allowed them to decrypt and forge Wi-Fi frames in a matter of minutes, according to the researchers.

Of course, methods to circumvent the Wi-Fi security protocols also can be developed and used, as shown in [34]–[36]. Typically, [34] tested the efficiency of a self-built phishing tool, written in Python. This tool would send an email with an attached file (i.e., a script) which, if run, would return all stored Wi-Fi passwords and related SSIDs stored on the victim’s machine to the attacker. Another well-known technique is the Man-in-the-Middle attack (or MITM), which consists of the attacker putting its machine in between the communication of the client and the AP to actively collect or modify Wi-Fi traffic. MITM can also lead to phishing attempts to gather the credentials. This method is thoroughly discussed and put into practice in [35] against wireless networks, and automated against Wi-Fi clients specifically in [36]. The first study analysed how Kali Linux (also called “Kali”), probably the most widely used OS to test the security of IT Systems, could be used to teach about MITM by describing the setup and launch of this attack. The second study developed a tool called *WifiMitm*, written in Python, which they successfully tested against 5 various environments (with variations relating to the laptop or smartphone model, OS, and Wi-Fi security protocol) which were compared. The MITM technique is one among many alternative penetration testing techniques that can be tested against Wi-Fi using Kali, which includes a default set of Wi-Fi hacking tools as summarised by [37]. In [37], they also tried to run some of the well-known Wi-Fi attacks that can be performed using Kali, notably the *aircrack-ng* suite that was mentioned previously to crack the Wi-Fi PSK after having captured the WPA/WPA2 handshake.

Giving a broader overview of the Wi-Fi attack paths, the study by Kohlios and Hayajneh [2] from 2018 is recommended for its completeness and quality. The authors present a comprehensive view of known Wi-Fi attacks, and the multiple attacking phases attackers have to go through. It provides a precise and structured vision of various attack pathways an attacker can perform, from WEP to WPA3 networks with an extensive overview of at-the-time available attack methods, such as the KRACK vulnerability.

When looking for studies specifically on KRACK, one research group also found the related vulnerability unintentionally. Indeed, the study led by Garbelini *et al* [38], describes how they developed an automated tool that tries various kinds of Wi-Fi tests. Their core technique was to use fuzzing to uncover non-compliant behaviour (or at least, unexpected responses within the Wi-Fi protocols). As the researchers explain, fuzzing is a technique that

aims at sending various values to an IT system to see how it reacts. The goal of this attack is usually to see how the IT system would handle malformed or unusual inputs. These non-compliant behaviours were either inconsistencies with the protocol specifications, crashes, or plain vulnerabilities. They tested their tool on real-world Wi-Fi-enabled devices, which allowed them to test Wi-Fi clients and AP implementations of the 802.11 standards. It discovered new Wi-Fi vulnerabilities, which were assigned common vulnerability and exposure (CVE) identifiers, and discovered KRACK as well, which was already known at the time.

Until now, different attacks targeting the PSK mode of WPA/WPA2 were discussed but the Enterprise mode also has shown to be vulnerable to attacks, if not implemented securely. For example, Abo-Soliman and Azer [39] compared the vulnerabilities affecting WPA2 in Enterprise mode and PSK mode. They noted that the password used for a certain username in an Enterprise mode can also be brute forced, as the usual Enterprise implementation is that the username is sent in plaintext (thus the attacker knows which user exists) and that the number of wrong authentications is usually unlimited. In [40], another research group performed an Evil Twin attack on a university campus WPA2 Enterprise network, meaning that they set up an AP that would broadcast itself as being the legitimate AP clients need to connect to. The interesting aspect of Evil Twin is that it doesn't necessarily need victims to interact with their "fake" AP: by broadcasting itself as the legitimate AP, with the "automated connection" feature enabled on devices, clients may connect directly to an AP without user interaction, thus providing the victim's credentials (used in an Enterprise system). The authors roamed the campus with their attacking system and estimated to have gathered around 7% of the credentials of the 2'700 users connected to the university network.

### **4.3 Assessing Wi-Fi Security through Wardriving**

Moving away from practical implementations of Wi-Fi attacks, a whole set of studies took an interest in estimating the Wi-Fi environment security level in their respective regions, using the previously mentioned wardriving method. The first study found on this topic was the analysis by Schreuders and Bhat in 2013 in Leeds (UK) titled "*Not All ISPs Equally Secure Home Users - An Empirical Study Comparing Wi-Fi Security Provided by UK ISP*" [41]. They aimed to compare the default security settings of the routers provided by major ISPs. They performed wardriving in town, collecting the broadcasted data of around 8'000 APs and compared the ISPs. As an example, they showed how one telecommunication

company still provided routers with WEP in almost 40% of the cases, and how another one set the default password of the provided routers to “sky”, a very easily guessable one. A meta-review of such studies was performed by a 2022 study containing a very thorough literature review of articles focusing on assessing the Wi-Fi security of geographical regions, mainly through wardriving [42]. This research team also performed wardriving to get a better overview of the Wi-Fi security level in a town located in North Cyprus. They collected data on more than 20’000 routers, among which around 50% used WPS, an easily hackable protocol. Overall, their results showed that while WEP was rarely used, some WLANs were using no security at all (25%). Also, they discovered that WPA/WPA2 (meaning both were supported) was used in 32% of cases, while most (43%) routers would use WPA2 only.

Another such analysis was [43], which performed wardriving in Riga (LV) in 2017. They also tested attacks on WEP, WPA/WPA2-PSK protocols, and the WPS security feature on various routers. Their aim was also to estimate the overall attack surface, by first gathering information on the security put in place on clients’ routers, and then testing specific attacks on selected routers. The data collected consisted of approximately 2’000 APs. Around 80% of them were supporting WPA2. Also, interesting to note is that while they gathered data with their tool, they also performed an online survey asking participants to indicate which Wi-Fi security protocol they were using for their routers: almost 50% responded that they didn’t know, while most (50%) shared that they have WPA or WPA2.

Other related studies were the analyses by Sebbar *et al* [44], Hossain *et al* [45], and Delija *et al* [46]. The research of Sebbar’s team took place in Rabat (MA) in 2016. They showed that almost 80% of routers had WPA/WPA2 enabled, and also did a brief analysis of the market share of the different ISPs. To do this, they looked at the broadcasted name of the APs, which often includes the ISP’s brand, allowing them to link the routers to the corresponding ISP. Unfortunately, the authors didn’t go further in comparing which ISP was implementing which security protocol. Hossain’s study in 2019 took place on a university campus in Dhaka (BD) to gather information about the Wi-Fi routers’ vulnerabilities to raise awareness of Wi-Fi security. What made their study, particularly novel is the use of *Nessus*, a known vulnerability scanner, on the Wi-Fi routers, which allowed the researchers to provide a list of detected vulnerabilities for the identified APs. The KRACK vulnerability wasn’t mentioned in the paper. Finally, another wardriving analysis was performed by Delija *et al* [46] in 2021 in Karlovac (HR) which determined that almost 90% of routers

were using WPA2 out of around 1'250 APs. This study also discussed the legal and ethical aspects of their methodology, which added interesting insights into these rarely discussed points in Wi-Fi security-related research papers.

## 5 Problem Statement

As suggested in the literature review, Wi-Fi security was and is still being investigated by research groups at many levels. At every step of the development of Wi-Fi security protocols, it was shown that new vulnerabilities were discovered, with tools to abuse them in their wake. Concerning KRACK, it seems that although the set of vulnerabilities is already 6 years old, no research paper presented a clear walkthrough on how to perform and test it, except on a theoretical level in the original paper written by Vanhoef and Piessens [5].

Overall, looking at ways to assess the Wi-Fi security environment, it seemed that research groups often preferred quantitative collecting methods such as wardriving on Wi-Fi APs, instead of checking if Wi-Fi-enabled devices were vulnerable to certain attacks such as KRACK. While allowing for an overview of the proportion of WPA/WPA2 in Wi-Fi networks, their methods couldn't detail if performing the KRACK attacks "in the wild" would be successful, and if so, how and which devices would be vulnerable. To analyse the Wi-Fi security environment, mostly wardriving was used, and often only to evaluate the Wi-Fi security protocols used on Wi-Fi routers, without actually testing the APs for various vulnerabilities.

The Czech Master's thesis [26], while thoroughly describing KRACK, only tested parts of the KRACK vulnerabilities. In addition, it was written 4 years ago and updating its results would be useful to assess the current relevance of KRACK. Furthermore, only one example of traffic capture was presented in the thesis, giving little for the reader to analyse in terms of results. Rewriting a set of detection or testing scripts is considered time inefficient: the author of KRACK already published scripts for them to be tested. Indeed, Mathy Vanhoef published a set of scripts to test if clients and APs are vulnerable to it on GitHub [27]. Given his expertise, it is assumed that these are the most effective ones publicly available, which is why they will be selected for testing devices. He also shared scripts to specifically perform the so-called MC-MITM technique, which can be accessed at [47]. Besides, he wrote a Proof-of-Concept script for the specific case of Linux-Android cases of a zero key reinstallation [48]. Other repositories were found based on the original one, such as [49] and [50]. The first one is a fork of Vanhoef's original repository focusing on all-zero key reinstallation, while the second one explains in general terms how to perform the MC-MITM attack with the all-zero key KRACK attack only (i.e., only against Linux hosts or Android smartphones). Given that already in 2019, only 1 out of 21 devices was affected by the all-

zero key reinstallation [26], this test is considered to be too exclusive to evaluate KRACK, even more 4 years later. Note that Vanhoef didn't make his attacking scripts used for his discoveries public. This was done on purpose, as stated on the official KRACK webpage [16]. Because of thesis time constraints but also because of related ethical questions, reverse-engineering scripts to perform it was discarded as a goal. Developing an attacking tool undoubtedly poses questions, such as the respect of the author's non-disclosure intent and the case of potential illegitimate use of such a tool. The only relevant scripts currently identified are either verifying if devices (APs and clients) are vulnerable or showing how to perform an MC-MITM attack.

Therefore, this Master's thesis aims at documenting the KRACK testing scripts of Vanhoef [27] on Wi-Fi-enabled devices. Following this documentation process, the main goal is to test them on a set of devices, to suggest a reassessment of KRACK's relevance. At the same time, with an "attacker's perspective", this thesis will show if vulnerable devices can be found, and if so, display which models (laptop, smartphone, and their OS), if any, were affected by the attacks. The thesis will present multiple Wi-Fi communication captures to support the findings.

The core research objective for this thesis is to estimate how prone Wi-Fi-enabled devices are vulnerable to KRACK. Thus, the following research questions are stated:

- Q1. Given the identified available GitHub repositories, is it possible to successfully scan Wi-Fi-enabled devices for the KRACK vulnerability?
- Q2. If that is the case, what are the struggles to perform the vulnerability test?
- Q3. What's the current relevance of KRACK given obtained results with this tool?

For this study, documenting the identified testing scripts will also be key, in order to make the analysis reproducible. This is considered as one of the important added values of this thesis, as no simple and clear walkthrough exists for testing KRACK, and understanding how the attack works and can be performed. Embedded in these research questions are the following hypotheses:

- H1. A tool to test KRACK attacks can be run successfully against Wi-Fi-enabled devices.
- H2. This tool only works with an alignment of specific conditions.
- H3. As KRACK was discovered in 2017, it will be difficult to find vulnerable devices.



## 6 Methods

This chapter provides a brief overview of the selected methods used to address the three research questions. First, to understand if Vanhoef’s official testing scripts [27] are efficient at detecting KRACK on devices, the installation of the GitHub repository’s scripts is performed, followed by a code analysis. A testing setup will be defined, with details on the required configuration. Then, to successfully test the scripts, the list of tested commands is also defined. Furthermore, a code analysis is conducted to understand the outputs of said commands. Moreover, the method for sampling the tested devices is defined.

All these steps will help answer the second research question, aiming at identifying the challenges of testing KRACK against Wi-Fi-enabled devices. It will also be possible to assess KRACK’s relevance (i.e., the third and last research question) after having analysed the results of the tested devices. The analysis will consist of running all designated commands against each device, the exact commands being described further in this chapter. To reduce false positive rates, if a test shows up as positive, it is run two more times. This means that all test-positive results will have to be tested three times before concluding it is vulnerable to the test. The model, OS type and OS version will be documented for each device. To have a stronger understanding of the results, network analysis is applied using Wireshark.

Now concerning the tests specifically, two scripts are at the heart of Vanhoef’s GitHub repository [27]: *krack-test-client.py* and *krack-ft-test.py*. The tests which can be performed using these scripts are listed by Vanhoef and summarised in [Table 2](#) (p. 41). The first column shows the script, with the command to run for each test. As can be seen, most of them are meant to test clients and not APs. The unique test for APs can be performed only if the AP supports 802.11r, as it targets the FT handshake. This is more often the case in corporate environments, as suggested by Vanhoef in [5]. For time and sampling relevancy reasons, the *krack-ft-test.py* script targeting APs implementing 802.11r will be discarded from this thesis’ scope, but is nonetheless included in the table for completeness.

The reason for discarding it is that to answer the third research question, it was decided that sampling, testing, and thus comparing tested devices would be more relevant for clients than for specific types of APs. Indeed, the population of clients is widespread and diverse, while the scarcer 802.11r APs will not help as much in assessing today’s KRACK relevance. An additional reason for this decision is that there was no access to 802.11r APs, making a potential sampling of such APs, at the time of writing, unfeasible.

Table 2 – Summary of Vanhoef’s seven tests for KRACK [27]

| <i>Script</i>                               | <i>Test No.</i> | <i>Full Command</i>                     | <i>Test Details</i>                                                                                                                   |
|---------------------------------------------|-----------------|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| krack-test-client.py<br>(to test a client)  | 1.              | krack-test-client.py                    | Resends encrypted EAPOL Message 3 to check for PTK reinstallation (CVE-2017-13077) and GTK reinstallation (CVE-2017-13078).           |
|                                             | 2.              | krack-test-client.py --tptk             | Same <sup>15</sup> as test 1 (no option), except that a forged message 1 is sent before retransmitting the encrypted EAPOL Message 3. |
|                                             | 3.              | krack-test-client.py --tptk-rand        | Same as test 2 (--tptk), except that the forged message 1 contains a random ANonce.                                                   |
|                                             | 4.              | krack-test-client.py --replay-broadcast | Checks if the client accepts replayed broadcast frames.                                                                               |
|                                             | 5.              | krack-test-client.py --gtkinit          | Checks if the client installs the GTK with the given Receive Sequence Counter (RSC) during the 4-way handshake.                       |
|                                             | 6.              | krack-test-client.py --group            | Checks if the GTK gets reinstalled during the group key handshake (CVE-2017-13080).                                                   |
|                                             | 7.              | krack-test-client.py --group --gtkinit  | Checks if the GTK is installed with the given Receive Sequence Counter (RSC) during the group key handshake.                          |
| krack-ft-test.py<br>(to test an 802.11r AP) |                 | krack-ft-test.py                        | Retransmits ReassoReq frame of the FT handshake to the AP to check for PTK reinstallation (CVE-2017-13082).                           |

<sup>15</sup> According to Vanhoef, this variant is important as some clients are only vulnerable to KRACK in the 4-way handshake when a forged EAPOL Message 1 is injected, before resending EAPOL Message 3. Vanhoef cites *wpa\_supplicant* v2.6 used on Linux-based OSs as an example.

For the sampling of devices, the selection process follows a convenience sampling process, where devices are selected by asking relatives, friends and coworkers to provide Wi-Fi-enabled devices (smartphones, computers, tablets) for testing. This is considered suitable as the end goal is to test as many different devices as possible (model, OS, and version). Part of the objective is also to showcase if, as a potential attacker, it would be feasible to find vulnerable devices in direct proximity. Indeed, gathering a higher number of devices to test (e.g., in a public place) would be far more complex for ethical reasons (i.e., as the permission of device owners would be needed) and technical reasons (i.e., as device owners need to manually connect to the test AP, and stay connected without any Internet connection). Having briefly analysed the main script (*krack-test-client.py*), it seems possible to test multiple devices at once, although it isn't stated as such, probably because of potential radio interference issues. Also, to check for which device (i.e., model and OS) the result is printed out, the MAC address of each device would still need to be manually registered. Finally, the exact phone models and OS versions are documented for this study, which could rapidly become tedious with a higher number of tested devices.

Regarding the KRACK vulnerability, the following assigned CVEs will be tested on clients, with dedicated commands:

- **CVE-2017-13077** (CVSSv3 score of 6.8, “Medium”): reinstallation of the PTK.
  - o Tested with: `krack-test-client.py`
    - Variations with additional options: `--tptk, --tptk-rand`
- **CVE-2017-13078** (CVSSv3 score of 5.3, “Medium”): reinstallation of the GTK in the 4-way handshake.
  - o Tested with: `krack-test-client.py`
    - Variations with additional options: `--tptk, --tptk-rand`
- **CVE-2017-13080** (CVSSv3 score of 5.3, “Medium”): reinstallation of the GTK in the group key handshake.
  - o Tested with: `krack-test-client.py --group`

Note that the `--replay-broadcast` and `--gtkinit` additional options don't have specific CVEs assigned to them but are used to detect malfunctioning Wi-Fi clients. Indeed, they still showcase common vulnerabilities (e.g., accepting replayed messages). Also, the tests only support CCMP, which means only WPA2. This is considered acceptable as it is the most widely observed Wi-Fi security protocol (see [7] as cited in [Introduction](#)).

The described methods used for this thesis have their limitations, the major ones described below:

- The age of the KRACK vulnerability:
  - It has been 6 years since the public disclosure of KRACK, which allowed for time to patch devices. This potentially resulted in a strong reduction in the number of vulnerable devices, which, although a positive sign, would reduce the analysis potential of this thesis.
- The representativity of sampling:
  - The selected sampling method doesn't allow for a representative overview of the type of devices that an attacker might encounter, which makes it impossible to generalise the results. Nevertheless, the representativity of the tests is not crucial as it is a case study, showcasing what an attacker could find "in the wild". Also, diversity is considered when selecting devices, meaning that the tests won't be applied to the same types of devices multiple times.
- The dependencies of the testing setup:
  - It might be the case that the KRACK scripts work well in certain test cases, such as against specific OSs or Wi-Fi client software, or running the tests with specific WNIC models. Indeed, the authors of the KRACK attacks showed that they don't get the scripts to work against all types of devices.
- The reliability of the script:
  - As warned by Vanhoef, some tests are not always reliable (e.g., the `--gtkinit` option). Wi-Fi traffic captures can be used to verify such results. Also, to understand the reliability of the script, an analysis of the code will help in providing potential answers to understand the obtain results.

Note that for grammatical reviewing, the Generative AI tool "Grammarly" was also used. For more details on the tool, see the official website at <https://www.grammarly.com/> (last accessed on the 8<sup>th</sup> of August 2023).

## 7 Results

This chapter begins with a subchapter presenting the procedure to install the testing setup for KRACK, followed by a subchapter on a code analysis of the tests performed. Finally, a summary of devices and related test results will be presented.

### 7.1 Setup Procedure

Starting with the testing setup, the selected host machine has Windows 10 with VirtualBox<sup>16</sup> installed, with a Kali Linux virtual machine<sup>17</sup> (VM) set up. This is following Vanhoef's advice on his GitHub repository: he only tested his scripts on Kali Linux. When inserting a WNIC in a USB port, it should be noted that to be successfully used in the VM on a Windows machine, it needs to be added as a USB device in the VM's settings (under "USB"). Also, as the tests are sensitive to background noise, it is advised to run them in a place with as little radio (Wi-Fi, Bluetooth) interference as possible.

Before any test execution, it was observed that only particular WNICs were compatible with Vanhoef's scripts. Indeed, he doesn't specify which work and which don't, but only shares with which WNICs his tests were successfully tested in the "Issues" section of his repository. This was the first (but consequent) struggle: to find a compatible WNIC. For example, it was found that the two WNICs selected beforehand for this analysis couldn't be worked with, after many various attempts to reverse-engineer the issues. The tested WNICs were the AWUS036ACH (FCCID 2AB8788121) by ALFA Network Inc. and the TL-WN822N (EU) v5.0 (FCCID TE7WN88NV4) by TP-Link Corporation. The former uses a RTL8812AU chipset, while the latter operates with a RTL8192EU chipset. To save time, a search was conducted online for the same Wi-Fi chipset as Vanhoef's (AR9271). As mentioned by Vanhoef himself<sup>18</sup>, it is now difficult to find them as it has been a long time since their release. This might be the reason why vendors don't provide them anymore as easily. In the end, a WNIC with the AR9271 chipset by Qualcomm Atheros was ordered online

---

<sup>16</sup> See <https://www.virtualbox.org/> (Accessed 08.08.2023).

<sup>17</sup> The pre-built image source can be retrieved from the official webpage of the Kali Linux project, available at <https://www.kali.org/get-kali/#kali-platforms> (Accessed 11.06.2023).

<sup>18</sup> This refers to a reply of Vanhoef to another user on the KRACK GitHub repository, publicly available at <https://github.com/vanhoefm/krackattacks-scripts/issues/94#issuecomment-1428601663>. He mentions that the following WNICs should work: WNICs with the RT5572 chipset, the TP-Link TL-WN722N and Techno-ethical N150 HGA (both using the AR9271 chipset), and finally WNICs with the RTL8188CUS chipset.

from a Chinese manufacturer, with no visible FCC ID on the outside of the hardware (see [Figure 16](#)).



Figure 16: Selected WNIC with Atheros AR9271 chipset

The subsequent procedure to install the scripts can be summarised in two steps. First, the required packages need to be installed on the VM, and the GitHub repository cloned locally. Secondly, commands need to be run to finalise the environment's configuration before each test session.

In the first step, installation files (*build.sh* and *pysetup.sh*) need to be run, and hardware encryption has to be disabled. The exact flow of commands is shown below:

```
sudo apt update

sudo apt install libnl-3-dev libnl-genl-3-dev pkg-config libssl-dev net-
tools git sysfsutils virtualenv

git clone https://github.com/vanhoefm/krackattacks-scripts.git

cd krackattacks-scripts/krackattack

./build.sh

./pysetup.sh

cd krackattack

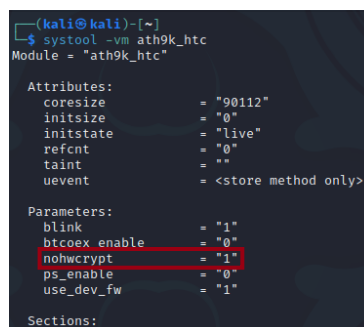
sudo ./disable-hwcrypto.sh
```

The *build.sh* script compiles a clean, modified *hostapd* daemon, written in C. *Hostapd* is a daemon used to create a Wi-Fi AP on Linux. It will run an AP, and tested devices will connect to it to undergo KRACK detection. The *pysetup.sh* file will create a virtual Python environment and read the *requirements.txt* file to install the correct Python packages and versions, namely *pycryptodome* (version 3.9.9) and *scapy* (version 2.4.4).

Hardware encryption must be deactivated as it might interfere with the scripts. Vanhoef mentions that it would be safer (in terms of reliability) to run the tests when transferring (“offloading”) the encryption/decryption operations directly to the software and not the hardware of the WNIC. In a comment inside of the *krack-test-client.py* code, he notes that “On some kernels, the virtual interface associated to the real AP interface will return frames with an already decrypted payload (this happens when hardware decryption is used)” (lines 116-117)<sup>19</sup>.

To reenale hardware encryption, one can run the *reenable-hwcrypto.sh* script provided in the repository, which will revert the options set for the defined WNICs, i.e., the “nohwcrypt=1” parameter in the newly created */etc/modprobe.d/nohwcrypt.conf* file will be removed. *Modprobe* is a utility used to load or offload modules of the Linux kernel. This is the path where Vanhoef adds a “nohwcrypt” (“no-hardware-encryption”) configuration for the selected WNIC. For optimal results, rebooting the VM is recommended.

Another extra step suggested by Vanhoef is to verify if the hardware encryption is indeed disabled, by checking that the nohwcrypt value is set to “1” when running the `systool -vm ath9k_htc` command<sup>20</sup> (see Figure 17). The option “-m” is used to specify a module, while “-v” is used to show all attributes and related values of the module. To verify that the *ath9k\_htc* module is installed, one can verify that it is in the output of the `lsmod` command when the WNIC is plugged in. The `lsmod` command is used for listing kernel modules.



```
(kali@kali)-[~]
$ systool -vm ath9k_htc
Module = "ath9k_htc"

Attributes:
  coresize      = "90112"
  initsize      = "0"
  initstate     = "live"
  refcnt        = "0"
  taint         = ""
  uevent        = "<store method only>"

Parameters:
  blink         = "1"
  btcoex_enable = "0"
  nohwcrypt     = "1"
  ps_enable     = "0"
  use_dev_fw    = "1"

Sections:
```

Figure 17: Hardware encryption is disabled on the *ath9k\_htc* kernel module

---

<sup>19</sup> Vanhoef explains this in more details when he tested his script with the *ath9k\_htc*: “With the *ath9k\_htc* it was not possible to detect the usage of an all-zero key when using hardware decryption. It seems as if the hardware tries decryption, thereby scrambling the context (N.B. “context”) of the frame, and then sees that decryption failed. This means userspace doesn’t receive the original frame, but a frame where the (encrypted) data is scrambled.” (Lines 336-340 of *krack-test-client.py*).

<sup>20</sup> Note that here, *ath9k\_htc* is the needed kernel module for the Atheros AR9271 WNIC chipset.

To identify which driver a WNIC different from the Atheros AR9271 different needs to communicate with the Linux kernel, the WNIC’s manufacturer and model need to be identified. Usually, the kernel will automatically load the corresponding module (or driver) when plugging in the WNIC. Besides, to check USB connectivity, the `lsusb` command can be run and the WNIC should be displayed. Finally, to check if the wireless interface is up, the `iwconfig` or `ip link` commands will confirm that a given interface name is activated (if not, running the `sudo ifconfig wlan0 up` command will activate it, ‘wlan0’ being the interface’s name in this example). In case of doubts, unplugging and re-plugging the WNIC will help identify the wireless interface name corresponding to the WNIC. Inside of the `./hostapd/hostapd.conf` (line 8), the wireless interface name must also be set.

After this first step, a second and final “configuration” step needs to be performed, each time before a test session. It consists of disabling Wi-Fi in the network manager (for example, using the `nmcli networking off` command), followed by the below commands:

```
sudo rfkill unblock wifi  
  
cd krackattack  
  
sudo su  
  
source venv/bin/activate
```

The first line will turn Wi-Fi transmissions on, while the following lines are used to enable the virtual Python environment needed for running the test scripts. This concludes the configuration. Now that the virtual environment is activated, we can run multiple tests one after the other on different client devices.

## 7.2 Script Analysis

This subchapter will give an overview of the reviewed script, i.e., *krack-test-client.py*, and the various options that can be tested against client devices. Before going into the details of the script, the way a test can be run is as follows: first, launch the *krack-test-client.py* script, then connect the Wi-Fi-enabled device to the Wi-Fi network named “testnetwork” and enter the password (“abcdefgh”). These are the default values defined in the `./hostapd/hostapd.conf` file (inside the `krackattacks-scripts` folder). The SSID is configured at line 88 and the PSK at line 1249 of the `hostapd.conf` file. It can also be seen at line 1240 that the



supported Wi-Fi security protocol is WPA2. After entering the correct password, a normal 4-way handshake between the client and the AP is performed. The test will start from there. The script bases its network packet handling capabilities on the well-known *scapy* Python package. Also, Vanhoef uses a personally developed Python package called *libwifi* (inspired by the C *libwifi* library), which handles Wi-Fi communications with the help of two other self-written libraries (*wifi.py* and *crypto.py*). Additionally, the *krack-test-client.py* script imports the local *wpaspy* package, used to control the operations performed on *hostapd*. Of course, other more common Python libraries like *socket*, *time*, *logging* or *sys* are also used in this script. The *logging* library is used by *scapy* to output messages, which make most of the printed messages during the test execution. To get the most of the logger's usage in the script, one can run the script with the `--debug` option to have access to more details on background operations. Additionally, the `-dd -K` option prints all debug outputs of *hostapd*, for extra-detailed information of all exchanged frames.

To summarise the script, 3 Python classes were written by Vanhoef, in addition to a few helper functions. These classes are **TestOptions**, **ClientState** and **KRAckAttackClient**. As the name reveals, the first **TestOptions** class retrieves the test options provided by the user. These are split into 2 categories:

1. the target or type of the attack (*ReplayBroadcast*, *ReplayUnicast*, *Fourway* which targets the PTK and GTK reinstallation in the 4-way handshake, and *Groupwh* which targets GTK reinstallation in the group key handshake),
2. and the variant of TPTK attack (if performed), explained further in the subchapter.

Note that the *ReplayUnicast* test option isn't used throughout the code, for unidentified reasons. By default, the *Fourway* option is selected, the TPTK variant set to 0 (*TptkNone*), and the additional *gtkinit* attribute set to "False". This last attribute checks if the provided RSC (Receive Sequence Counter) value used when installing the GTK is used (which is good), or if the client resets it to 0 (which would be bad) during either the 4-way handshake or the group key handshake.

The **ClientState** class differentiates the test state and the client state, which can be "UNKNOWN", "VULNERABLE", or "PATCHED", and keeps track of changes in the client state depending on the results of the tests. Multiple class methods are needed to either assess if the client is vulnerable or patched, or to just set the encryption key. To verify if a device is vulnerable, class methods are used to identify IV/nonce reuse. This is done by setting an

`IvCollection` object (imported from the *libwifi* library, more specifically the *wifi.py* library), and then using class methods to identify and track any IV (nonce) reuse. Note that in this chapter, the terms “IV” and “nonce” will be used interchangeably. Remember also that the PTK and the GTK have separate IV incrementations, meaning that if multiple unicast frames were exchanged (thus incrementing the IV linked to the PTK), it won’t affect the IV used for the next multicast/broadcast frame. Indeed, the RSC is the starting packet number (IV) for the GTK, which gets incremented for each encrypted multicast/broadcast frame sent. The most important class method is probably `check_pairwise_reinstall()`. In this method, multiple PTK reinstallations will be waited for before assuming that the client is patched<sup>21</sup>, while the actual PTK reinstallation occurs in the `KRAckAttackClient` class. To check for a group key reinstallation from the client, the AP will send replayed broadcast ARP (Address Resolution Protocol) requests on the network, and check if clients will respond to it. Even though the GTK is not used to encrypt frames by the client as mentioned earlier, if it replies to a replayed encrypted ARP request (i.e., encrypted with the GTK and reusing a GTK-IV), it means that the client accepted the frame. It is also in the `ClientState` class that various class methods for group key reinstallation detection reside.

The class running the simulated attack is the **KRAckAttackClient** class. In its constructor class method, the *hostapd* configuration will be read to get the WNIC’s interface name. Also, the WNIC’s MAC address is grabbed using *scapy*. The main class method is `run()`, used to launch the test (line 463), as can be seen in the last line of the script (line 663) inside of the `main()` function. It starts by setting the options used for the test by declaring a `TestOptions` class. Furthermore, a `Ctrl` object is declared to communicate with *hostapd* (using the *wpa* library mentioned earlier) at line 481. The following lines are used for further AP configuration, like having the *scapy* library handle Dynamic Host Configuration Protocol (DHCP) requests which will be sent by the connecting clients to get their IP address<sup>22</sup>.

After having finished setting up the AP, the actual testing can be performed. This is done through an endless loop, which needs to be terminated manually. Each loop can be

---

<sup>21</sup> As mentioned by Vanhoef [27], it could be that frames are lost due to important background noise, and thus the client state remains unknown. In practice, the script checks for 5 “IV reuse cycles”, where a cycle takes longer each round (i.e., the time interval is doubled each round before resending EAPOL Message 3).

<sup>22</sup> This is the assumed reason why Vanhoef insists that for the tests to be successful, clients need to request their IP through DHCP [27]. In our tests, it was observed that by default, smartphones have this configuration enabled.

represented as a cycle of actions. Per cycle, an ARP request will be sent with the tested device's MAC address, looking for a GTK reinstallation (which could have occurred in the 4-way handshake or the group key handshake). Depending on the options used, the script will either perform a new 4-way handshake and track PTK-IVs or launch a new group key handshake and test if a replayed ARP request (i.e., where a GTK-IV was reused) gets a response. The Appendix (section II) can be consulted to understand the general cycle for each test. More details on the process followed by the script are given now.

First, if the *gtkinit* attribute is set, should it be when testing the 4-way handshake or the group key handshake, then the script will start by making *hostapd* reinitialise the GTK. The commands used to communicate with the *hostapd* daemon are simple to read but not always straightforward for interpretation, as can be seen in the below example (line 531).

```
hostapd_command(self.hostapd_ctrl, "RENEW_PTK " + client.mac)
```

Here, it seems to be renewing the PTK, while in reality, it is asking *hostapd* to restart a new handshake with a specific client, using its MAC address. This is assumed to originate from the *ctrl\_iface.c* file contained in the *hostapd* folder. For the GTK renewal, a similar syntax is used, as shown in the below example. Note that since the GTK renewal is unilaterally performed by the AP, the command below doesn't imply going through another 4-way handshake (line 520), nor a group key handshake. The latter will only be used to share the new GTK with clients.

```
hostapd_command(self.hostapd_ctrl, "RENEW GTK")
```

If the *gtkinit* option isn't used, the script will first reinitialise the RSC value (i.e., the counter for GTK-IVs) of the current GTK. Then, it will look at the selected TPTK option:

- if the `--tptk` option was used, then EAPOL Message 1 is replayed (with the same ANonce as before), followed by a replayed EAPOL Message 3,
- if the `--tptk-rand` option was used, then EAPOL Message 1 is sent again (with a new ANonce), followed by a replayed EAPOL Message 3,
- if no additional option was used, only EAPOL Message 3 is replayed.

The reason for sending EAPOL Message 1 before was documented in an online Addendum of Vanhoef and Piessens [16], and cited by Vanhoef again in a follow-up study on KRACK [25]. According to their research, *wpa\_supplicant* v2.6 (and other clients) are only vulnerable to KRACK if an EAPOL Message 1 is replayed (or sent again with a new ANonce) before retransmitting EAPOL Message 3. In all cases, after EAPOL Message 3 is sent again,

the script expects an encrypted EAPOL Message 4 from the client. This is the frame in which IV reuse is analysed. Since the vulnerable client accepts this new EAPOL Message 3, it should also reinitialise its IV, leading to PTK-IV reuse. The script keeps track of all IVs used. As stated before, when using the `--debug` option, a user can follow all “debug-level” log messages of the script. In such a case, the tester can see which PTK-IV was used (for which frame, identified by a sequence number).

Following the transmitted EAPOL message(s), the script will then check for the client’s behaviour concerning replayed ARP requests (checking for group key reinstallation). This is done by using the `broadcast_check_replies()` method (of the `ClientState` class). If the `ClientState` is not in the “VULNERABLE” state, it will use the `broadcast_send_request()` method (of the `KRAckAttackClient` class) to send the ARP requests. In this method, if the `--group` option was used, a new group key handshake message will first be sent, i.e., an EAPOL “group” message 1. In any case, the method will then replay a broadcast ARP request with the client’s MAC address, looking for an ARP response. As mentioned in “[The KRACK Vulnerability](#)” subchapter (3.3), a GTK reinstallation only allows an attacker to replay broadcast/multicast frames. If the client is vulnerable, after having reinstalled the GTK (either through the 4-way handshake or the group key handshake), it will reset its RSC counter. When receiving a replayed ARP request, it will accept the GTK-IV used and reply to the request. In the script, at least 5 out of all replayed ARP requests needs to be replied to state that the client is vulnerable. If the client is patched, it should ignore the requests.

### 7.3 Devices tested for KRACK Vulnerability

In total, 29 Wi-Fi-enabled devices were analysed. [Table 3](#) (see pages 53 to 54) presents the overview of tested devices with the seven tests (options) of the `krack-test-client.py` script. This sums up to 203 tests performed in total. As can be seen, computers, tablets, and mostly smartphones were analysed. The main goal was to test a variety of hardware and OSs, and document as thoroughly as possible the models and OS versions (such as the kernel version) which was overall successful. Some device owners didn’t have a lot of time for sharing their personal smartphone, which made this documentation sometimes less comprehensive. In any case, the most important data could be retrieved.

The results of the tests can be seen in [Table 3](#), with a colour legend summarising the results: a device should either be vulnerable (**red**) or not vulnerable (**green**). In practice, it was observed that some tests were unsuccessful (**yellow**), usually because the client would

disconnect and reconnect after a short period of time in the course of testing. This might be a security feature that prevents a device from staying connected to an AP sending unexpected frames. “N.A.” (grey) indicates that for unknown reasons, the script would run forever without printing out a result. Finally, “Possibly Vuln.” (orange) refers to the fact that if the device accepts replayed broadcast frames, then the group-related tests (i.e., Tests 5, 6 and 7, see below) are considered unreliable. This is stated on the GitHub repository of the tests: *“If the client accepts replayed broadcast frames, this must be patched first. If you do not patch the client, our script will not be able to determine if the group key is being reinstalled (because then the script will always say the group key is being reinstalled).”* [27]

As a reminder for [Table 3](#) (pages 53 to 54), the test number represents the command run:

- Test 1: `krack-test-client.py`
- Test 2: `krack-test-client.py --replay-broadcast`
- Test 3: `krack-test-client.py --tpk`
- Test 4: `krack-test-client.py --tpk-rand`
- Test 5: `krack-test-client.py --gtkinit`
- Test 6: `krack-test-client.py --group`
- Test 7: `krack-test-client.py --group -gtkinit`

Tests 1, 3 and 4 check both PTK (CVE-2017-13077) and GTK (CVE-2017-13078) reinstallation in the 4-way handshake. Therefore, the colour code for the result applies for both PTK and GTK, unless specified otherwise. Test 6 verifies if the device was vulnerable to CVE-2017-13080. In total, only 2 devices (devices no. 3 and 4) were test-positive to CVE-2017-13077, respectively the BlackBerry device and the Samsung S5 Plus. This means that the tests state that the devices are reinstalling the PTK when an EAPOL Message 3 is sent again. The BlackBerry and Samsung S5 Plus are also vulnerable when a new EAPOL Message 1 is transmitted first with a random ANonce (Test 3), followed by a transmission of a new encrypted EAPOL Message 3. Also, only 3 devices were positive for CVE-2017-13080. Other test-positive results for Tests 2, 5 and 7 were also detected and will be discussed in the next chapter.

Table 3: Devices tested for KRACK ( $n = 29$ )

|     | Not Vuln.                               |  | Vuln. |  | Possibly Vuln.                    |  | Test Unsuccessful |  | N.A. | Tests |   |    |   |   |   |   |
|-----|-----------------------------------------|--|-------|--|-----------------------------------|--|-------------------|--|------|-------|---|----|---|---|---|---|
| No. | Device Model                            |  |       |  | OS version                        |  |                   |  |      | 1     | 2 | 3  | 4 | 5 | 6 | 7 |
| 1.  | Surface 3                               |  |       |  | Win10 (build 10.0.19045)          |  |                   |  |      |       |   |    |   |   |   |   |
| 2.  | Dell Inc Latitude E7450                 |  |       |  | Ubuntu 22.04.2 LTS                |  |                   |  |      |       |   |    |   |   |   |   |
| 3.  | Blackberry Bold 9700                    |  |       |  | BlackBerry OS v6.0 Bundle 2949    |  |                   |  |      | 23    |   | 24 |   |   |   |   |
| 4.  | Samsung Galaxy S5 Plus (Model SM-G901F) |  |       |  | Android 6.0.1                     |  |                   |  |      |       |   |    |   |   |   |   |
| 5.  | Samsung Galaxy S6 Edge (Model SM-G928F) |  |       |  | Android 7.0                       |  |                   |  |      |       |   |    |   |   |   |   |
| 6.  | Samsung Galaxy Note 9 (Model SM-N960F)  |  |       |  | Android 10                        |  |                   |  |      |       |   |    |   |   |   |   |
| 7.  | Xiaomi Redmi Note 9                     |  |       |  | Android 10                        |  |                   |  |      |       |   |    |   |   |   |   |
| 8.  | Asus ZenFone 6                          |  |       |  | Android 10                        |  |                   |  |      |       |   |    |   |   |   |   |
| 9.  | Xiaomi Mi A3 (Model M1906F9SH)          |  |       |  | Android 11                        |  |                   |  |      |       |   |    |   |   |   |   |
| 10. | Samsung Galaxy S10 (Model SM-G973U)     |  |       |  | Android 11                        |  |                   |  |      |       |   |    |   |   |   |   |
| 11. | OnePlus 6 (Model A6003)                 |  |       |  | OxygenOS v11.1.2.2 (Android 11)   |  |                   |  |      |       |   |    |   |   |   |   |
| 12. | Fairphone 4 (FP4)                       |  |       |  | Android 12                        |  |                   |  |      |       |   |    |   |   |   |   |
| 13. | Vivo Model V2023                        |  |       |  | Funtouch OS12 Global (Android 12) |  |                   |  |      |       |   |    |   |   |   |   |
| 14. | Samsung Galaxy S20 FE 5G                |  |       |  | Android 13                        |  |                   |  |      |       |   |    |   |   |   |   |
| 15. | Samsung Galaxy S23+                     |  |       |  | Android 13                        |  |                   |  |      |       |   |    |   |   |   |   |
| 16. | Sony Xperia 1 Mark 3                    |  |       |  | Android 13                        |  |                   |  |      |       |   |    |   |   |   |   |
| 17. | Oppo Find X5 Pro                        |  |       |  | ColorOs (Android 13)              |  |                   |  |      |       |   |    |   |   |   |   |
| 18. | Oppo Reno 8 Pro 5G                      |  |       |  | ColorOs (Android 13)              |  |                   |  |      |       |   |    |   |   |   |   |
| 19. | MacBook Air (Early 2015)                |  |       |  | macsOS Catalina (v10.15.7)        |  |                   |  |      |       |   |    |   |   |   |   |
| 20. | iPod Touch 1 (Model MC547LL/A)          |  |       |  | iOS 6.1.6                         |  |                   |  |      |       |   |    |   |   |   |   |
| 21. | iPad 2 (Model MC982FD/A)                |  |       |  | iOS 9.3.1                         |  |                   |  |      |       |   |    |   |   |   |   |
| 22. | iPad 2 (Model MD366LL/A)                |  |       |  | iOS 9.3.5                         |  |                   |  |      |       |   |    |   |   |   |   |
| 23. | iPhone 5                                |  |       |  | iOS 10.3.4                        |  |                   |  |      |       |   |    |   |   |   |   |
| 24. | iPhone 5S (Model ME432KN/A)             |  |       |  | iOS 11.4.1                        |  |                   |  |      |       |   |    |   |   |   |   |
| 25. | iPhone SE (Model MP8822FD/A)            |  |       |  | iOS 13.4                          |  |                   |  |      |       |   |    |   |   |   |   |

<sup>23</sup> Test 1's output states that the PTK was reinstalled but not the GTK.<sup>24</sup> Test 3's output states that the PTK was reinstalled but not the GTK.

|                          | Not Vuln.          |  | Vuln. |  | Possibly Vuln. |  | Test Unsuccessful |  | N.A. | Tests |   |   |   |    |   |   |
|--------------------------|--------------------|--|-------|--|----------------|--|-------------------|--|------|-------|---|---|---|----|---|---|
| No.                      | Device Model       |  |       |  | OS version     |  |                   |  |      | 1     | 2 | 3 | 4 | 5  | 6 | 7 |
| 26.                      | iPhone 14 Pro      |  |       |  | iOS 16.5.1     |  |                   |  |      |       |   |   |   |    |   |   |
| 27.                      | iPhone XR          |  |       |  | iOS 16.5       |  |                   |  |      |       |   |   |   |    |   |   |
| 28.                      | iPhone 12          |  |       |  | iOS 16.5.1     |  |                   |  |      |       |   |   |   |    |   |   |
| 29.                      | iPad Air (5th Gen) |  |       |  | iPadOS 16.2    |  |                   |  |      |       |   |   |   |    |   |   |
| Total of "Vuln." results |                    |  |       |  |                |  |                   |  |      | 2     | 1 | 2 | 0 | 13 | 3 | 6 |

(Table 3: Devices tested for KRACK ( $n = 29$ ))

To verify the results and understand what is happening in the background, Wireshark was used to capture transmissions with specific devices. Since the testing script already creates a virtual monitoring interface (in this case, called *monwlan0*), it can be used on Wireshark to detect IV reuse (and replies to ARP frames reusing GTK-IVs). The CCMP Header is where the IV can be found (i.e., the PN). Before detecting IV reuse, the CCMP header containing the IV needs to be briefly understood first (see [Figure 18](#)). In Wireshark, it can be found in encrypted frames under the "CCMP Header" field, as will be shown later.

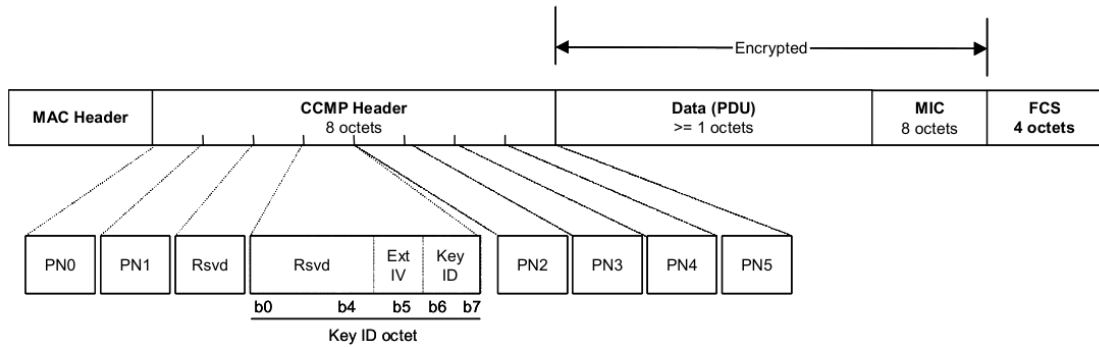


Figure 18: CCMP Header [13, p. 180]

The CCMP header contains the IV, split into 6 bytes (from PN0, the least significant byte or LSB, to PN5, the most significant byte or MSB). Note that just like the PN, the *KeyID* byte must be reordered from the MSB to LSB (from bit 7 to bit 0). To complete the CCMP header, the "Rsvd" (i.e., "Reserved" for future use) parts are always '0' bits, the "ExtIV" value is always a '1' bit for CCMP-encrypted payloads, and the "KeyID" value will identify one of four possible encryption keys used.

Now, using Wireshark, the CCMP Header field can be inspected. It will appear when a Wi-Fi frame is encrypted and used to identify IV reuse (Figure 19). While running the *krack-test-client.py* script, the *monwlan0* virtual interface is launched is selected when capturing transmissions on Wireshark.

| No. | Time        | Source            | Destination       | Protocol | Length | Replay Counter | CCMP Ext. Initialization Vector | Key Index | Info                                                    |
|-----|-------------|-------------------|-------------------|----------|--------|----------------|---------------------------------|-----------|---------------------------------------------------------|
| 159 | 3.014150017 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 146    | 1              |                                 |           | Key (Message 1 of 4)                                    |
| 161 | 3.026031524 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 195    | 1              |                                 |           | Key (Message 2 of 4)                                    |
| 163 | 3.029571288 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 202    | 2              |                                 |           | Key (Message 3 of 4)                                    |
| 164 | 3.035435538 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 2              |                                 |           | Key (Message 4 of 4)                                    |
| 167 | 3.084756927 | Shenzhen_2f:f6:f0 | Broadcast         | 802.11   | 152    |                |                                 |           | Beacon frame, SN=0, FN=0, Flags=....., BI=100, SS       |
| 168 | 3.093198093 | ::                | ff02::16          | ICMPv6   | 286    |                | 0x000000000001                  |           | 0 Multicast Listener Report Message v2                  |
| 169 | 3.097775130 | ::                | ff02::16          | ICMPv6   | 257    |                | 0x000000000001                  |           | 1 Multicast Listener Report Message v2                  |
| 171 | 3.133263841 | 0.0.0.0           | 255.255.255.255   | DHCP     | 412    |                | 0x000000000002                  |           | 0 DHCP Request - Transaction ID 0x149ea92f              |
| 172 | 3.140157206 | 0.0.0.0           | 255.255.255.255   | DHCP     | 383    |                | 0x000000000002                  |           | 1 DHCP Request - Transaction ID 0x149ea92f              |
| 174 | 3.144947894 | 192.168.100.254   | 192.168.100.2     | DHCP     | 394    |                | 0x000000000001                  |           | 0 DHCP ACK - Transaction ID 0x149ea92f                  |
| 177 | 3.187073279 | Shenzhen_2f:f6:f0 | Broadcast         | 802.11   | 152    |                |                                 |           | Beacon frame, SN=0, FN=0, Flags=....., BI=100, SS       |
| 178 | 3.203917609 | 22:61:71:23:de:e6 | Broadcast         | ARP      | 118    |                | 0x000000000003                  |           | 0 Who has 192.168.100.254? Tell 192.168.100.2           |
| 179 | 3.208293934 | 22:61:71:23:de:e6 | Broadcast         | ARP      | 89     |                | 0x000000000003                  |           | 1 Who has 192.168.100.254? Tell 192.168.100.2           |
| 180 | 3.210525295 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | 802.11   | 66     |                |                                 |           | QoS Null function (No data), SN=982, FN=0, Flags=...    |
| 182 | 3.211073781 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | ARP      | 91     |                | 0x000000000002                  |           | 0 192.168.100.254 is at c0:1c:30:2f:f6:f0               |
| 183 | 3.213548881 | 192.168.100.2     | 192.168.100.254   | TCP      | 150    |                | 0x000000000004                  |           | 0 45938 → 853 [SYN] Seq=0 Win=65535 Len=0 MSS=1220 SACK |

Figure 19: Wireshark capture with CCMP Header column with device no. 12 (start of Test 1)

In Figure 19, a portion of the capture between the Fairphone FP4 (device no. 12) and the Atheros AR9271 (used by *hostapd*) is displayed while running Test 1. This device was not vulnerable to the test. The displayed MAC address of the AP starts with “Shenzhen”. The view is filtered for the AP’s and the Fairphone’s MAC addresses. Note that the display filters (if used) for each presented Wireshark capture can be consulted in the Appendix (section I). Sometimes, a local IPv4 address is also shown instead of the MAC address, such as “192.168.100.254” (which would be the AP), or “192.168.100.2” (i.e., the client). More rarely, IPv6 addresses are used, such as in frame no. 168. Also noteworthy is that EAPOL Message 4’s were not encrypted for this device, although it was suggested in [5] that this often happens.

The starting 4-way handshake is performed, with the EAPOL replay counter values shown under the “Replay Counter” column. The “CCMP Ext. Initialization Vector” column represents the IV value of the frame, extracted from the CCMP Header. Following the 4-way handshake, 802.11 Data frames are encrypted either using the GTK, or the TK. The “Key Index” column indicates this: a *Key Index* value of “0” means that the TK was used, while “1” implies that the GTK was used. The case of the client encrypting a unicast frame (using the TK) and sending it to the AP, for it to re-encrypt it (using the GTK) and send it on the WLAN, can be seen, such as in frames no. 168-169. To understand such procedures, the CCMP Header needs to be deconstructed further. Indeed, the IV value contained in the CCMP Header can be extracted, to identify the IVs that are used by the client and the AP. Remember that the IVs are incremented for each use of the TK and GTK separately.



|     |                                                                                       |      |                         |                               |                        |                    |                                        |
|-----|---------------------------------------------------------------------------------------|------|-------------------------|-------------------------------|------------------------|--------------------|----------------------------------------|
| 168 | 3.093198093                                                                           | ::   | ff02::16                | ICMPv6                        | 286                    | 0x0000000000000001 | 0 Multicast Listener Report Message v2 |
|     |                                                                                       |      |                         |                               |                        |                    |                                        |
| >   | Frame 168: 286 bytes on wire (2288 bits), 286 bytes captured (2288 bits) on interface | 0000 | 00 00 24 00 2f 40 00 a0 | 20 08 00 00 00 00 00 00       | ..\$/@. ....           |                    |                                        |
| >   | Radiotap Header v0, Length 36                                                         | 0010 | 99 bc f2 43 00 00 00 00 | 10 30 6c 09 c0 00 dd 00       | ...C... ..01....       |                    |                                        |
| >   | 802.11 radio information                                                              | 0020 | 00 00 dd 00 88 41 2c 00 | c0 1c 30 2f f6 f0 22 61       | ...A... ..0/..a        |                    |                                        |
| >   | IEEE 802.11 QoS Data, Flags: .p....TC                                                 | 0030 | 71 23 de e6 33 33 00 00 | 00 16 00 00 00 00 01 00       | q#..33.....            |                    |                                        |
| >   | Type/Subtype: QoS Data (0x0028)                                                       | 0040 | 00 20 00 00 00 00 00 00 | 38 76 e7 78 45 7f 08 97 72 0c | ....8v..xE...r..       |                    |                                        |
| >   | Frame Control Field: 0x8841                                                           | 0050 | c1 b8 3f eb 3f c0 f3 5c | 0e b3 a7 28 82 f2 75 6b       | ..?.. \ ..(..uk        |                    |                                        |
| >   | .000 0000 0010 1100 = Duration: 44 microseconds                                       | 0060 | 00 c7 e6 24 1e 21 18 48 | 53 0a 45 7f 4e 7b b9 50       | ...\$.!..H S..E..N{..P |                    |                                        |
| >   | Receiver address: Shenzhen_2f:f6:f0 (c0:1c:30:2f:f6:f0)                               | 0070 | e0 e8 e1 0c 43 5e 9e 4c | c5 2f 09 35 40 75 c0 c3       | ...C^..L ..-/5@u..     |                    |                                        |
| >   | Transmitter address: 22:61:71:23:de:e6 (22:61:71:23:de:e6)                            | 0080 | e4 cd cf c6 41 b0 7f 48 | dd a8 68 00 4b 5e 7e c5       | ...\$n..U... ..s..a3.. |                    |                                        |
| >   | Destination address: IPv6mcast_16 (33:33:00:00:00:16)                                 | 0090 | f6 bb 24 6e a5 55 9c 82 | eb 19 73 db fe 61 33 07       | 'Ez.... ..E...g..      |                    |                                        |
| >   | Source address: 22:61:71:23:de:e6 (22:61:71:23:de:e6)                                 | 00a0 | 27 45 7a b4 a3 15 f8 b0 | 83 45 13 d4 2e 67 af c6       | ..H4.(... ..i...X..    |                    |                                        |
| >   | BSS Id: Shenzhen_2f:f6:f0 (c0:1c:30:2f:f6:f0)                                         | 00b0 | c1 48 34 ef 28 90 1f f1 | ee b7 69 1c 9b 58 1f 3a       | }I>... ..PiU           |                    |                                        |
| >   | STA address: 22:61:71:23:de:e6 (22:61:71:23:de:e6)                                    | 00c0 | 7d 49 3e e9 31 ff 94 ca | ca 1b bb 8b 1b 50 69 55       | ..R..Y^4.. ..X.. ..c   |                    |                                        |
| >   | .... .. 0000 = Fragment number: 0                                                     | 00d0 | d9 52 c5 0a 59 5e 34 cb | eb 58 c6 7c cb c4 5c 63       | H..[T..U... ..m..V..   |                    |                                        |
| >   | 0000 0000 0000 .... = Sequence number: 0                                              | 00e0 | 48 ae 07 5b 54 c0 55 c0 | b8 3a 88 a1 6d bf 56 81       | Yv..%..y.. ..m..V..    |                    |                                        |
| >   | Frame check sequence: 0xbace599f [unverified]                                         | 00f0 | 59 76 91 25 8b 99 79 f0 | a6 08 12 80 8e d5 04 60       | ..*G.. ..l.. ..js0...  |                    |                                        |
| >   | [FCS Status: Unverified]                                                              | 0100 | bf 2a 47 90 20 ae 6c a5 | a7 0a 4a 73 4f dd 16 c6       | ..0..S..v... ..Y..     |                    |                                        |
| >   | QoS Control: 0x0000                                                                   | 0110 | 9a 30 86 02 53 01 76 a6 | 05 c0 9f 59 ce ba             |                        |                    |                                        |
| >   | CCMP parameters                                                                       |      |                         |                               |                        |                    |                                        |
| >   | CCMP Ext. Initialization Vector: 0x000000000001                                       |      |                         |                               |                        |                    |                                        |
| >   | Key Index: 0                                                                          |      |                         |                               |                        |                    |                                        |
| >   | [TK: 60cc9d063a24a13f780f831d92db9088]                                                |      |                         |                               |                        |                    |                                        |
| >   | [PMK: e801548d2e7e7d6cfaddc2345e5be68167759161102a15a864540e6e59796410]               |      |                         |                               |                        |                    |                                        |
| >   | Logical-Link Control                                                                  |      |                         |                               |                        |                    |                                        |
| >   | Internet Protocol Version 6, Src: ::, Dst: ff02::16                                   |      |                         |                               |                        |                    |                                        |
| >   | Internet Control Message Protocol v6                                                  |      |                         |                               |                        |                    |                                        |

Figure 20: Detailed look at the CCMP Header of frame no. 168

In Figure 20, the CCMP Header of frame no. 168 is deconstructed to understand how the real value of the IV is determined. Here, the CCMP Header is “01 00 00 20 00 00 00 00” (in hexadecimal), with the PN bytes in green, the Reserved byte in purple and the KeyID byte in red. Joining the 6 bytes of the packet number (i.e., from PN5 to PN0) gives us the following bytes for the IV (in hexadecimal): “00 00 00 00 00 01”, which corresponds to “1” in decimal.

The TK and PMK values are also indicated in Figure 20 only because the Wi-Fi network password was entered in Wireshark to allow frame decryption, to show which key was used for CCMP encryption. These keys are never transmitted on the channel between the client and the AP. The ExtIV and the KeyID values can also be extracted from the 4<sup>th</sup> byte of the CCMP Header (“20” in hexadecimal). For frame no. 168, the following values are extracted: “20” (in hexadecimal) becomes “0010 0000” (in bits), therefore, the KeyID equals “00” (bits 7 and 6) and the ExtIV is “1” (bit 5). The KeyID indicates that the TK was used and the ExtIV is “1”, as defined in the 802.11 standard [13].

Now that the collected results and the IV identification were presented, further discussion on the obtained results is provided in the next chapter.

## 8 Discussion

This chapter discusses the obtained results and goes deeper into the analyses to assess their relevance. As a first general observation, the results showed that it was difficult to find Wi-Fi-enabled devices vulnerable to the core of KRACK, meaning PTK and GTK reinstallations (i.e., Test 1 and Test 6). Only older devices like the BlackBerry or Samsung S5 were positive to key reinstallations in the 4-way handshake. The OS and its version appear to be key elements in identifying vulnerable devices, while the year of device release doesn't seem to play a role. This is overall an encouraging outcome for Wi-Fi users. Nevertheless, it was found that devices reacted in different ways to the tests, should it be positive (i.e., vulnerable) or not. The chapter will be split into subchapters, starting with a general discussion of the obtained results, followed by a comparison between an overall non-vulnerable device and devices which tested positive for some of the commands. The chapter will be completed by a broader discussion including the research questions, as well as the impact and limits of this thesis' results.

### 8.1 General Observations

As mentioned earlier, there was a variety of responses from the devices being tested. For example, some smartphones would rapidly produce a pop-up message stating that no Internet connection was detected. This might be a security measure to protect against similar MITM attacks. Typically, the Sony Xperia smartphone (device no. 16) would show this pop-up message, and automatically disconnect by default if the user didn't confirm the connection after a few seconds. The manual confirmation of the user was needed to keep the smartphone connected to the network. In contrast, some devices would reconnect automatically to the network between 2 tests.

Test 2 came out positive only once, namely for a Windows 10 device. This will be investigated further in this chapter. No other device accepted replayed broadcast frames. Moving to the variants of the "basic" KRACK attack, Test 3 was positive only for the devices which were already vulnerable to Test 1, i.e., the Blackberry and Samsung S5 devices. The only test which was mostly inconclusive was Test 4. Indeed, it was observed that several smartphones would disconnect automatically after some time when testing the KRACK variant performed with the `--tpk-rand` option.

| No.  | Time         | Source            | Destination       | Protocol  | Length | Replay Counter | COMP Ext.        | Initialization Vector | Key Index | Info                                                                 |
|------|--------------|-------------------|-------------------|-----------|--------|----------------|------------------|-----------------------|-----------|----------------------------------------------------------------------|
| 2453 | 21.871991313 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 145    |                |                  |                       |           | Probe Request, SN=2772, FN=0, Flags=.....C, SSID="testnetwork"       |
| 2455 | 21.808615590 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 146    |                |                  |                       |           | Probe Response, SN=20, FN=0, Flags=....., BI=100, SSID="testnetwork" |
| 2456 | 21.881312572 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 70     |                |                  |                       |           | Authentication, SN=2773, FN=0, Flags=.....C                          |
| 2458 | 21.884457289 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 43     |                |                  |                       |           | Authentication, SN=21, FN=0, Flags=.....                             |
| 2459 | 21.886329695 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 163    |                |                  |                       |           | Association Request, SN=2774, FN=0, Flags=.....C, SSID="testnetwork" |
| 2462 | 21.904050232 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 95     |                |                  |                       |           | Association Response, SN=22, FN=0, Flags=.....                       |
| 2464 | 21.906185518 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 146    | 1              |                  |                       |           | Key (Message 1 of 4)                                                 |
| 2466 | 21.922491345 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 195    | 1              |                  |                       |           | Key (Message 2 of 4)                                                 |
| 2467 | 21.925069610 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 202    | 2              |                  |                       |           | Key (Message 3 of 4)                                                 |
| 2469 | 21.927620994 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 173    | 2              |                  |                       |           | Key (Message 4 of 4)                                                 |
| 2470 | 21.932237148 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | Broadcast | 802.11 | 152            |                  |                       |           | Beacon frame, SN=0, FN=0, Flags=....., BI=100, SSID="testnetwork"    |
| 2471 | 21.98465839  | ::                | ff02::16          | ICMPv6    | 366    |                | 0x00000000000001 |                       | 0         | Multicast Listener Report Message v2                                 |
| 2472 | 21.990641891 | ::                | ff02::16          | ICMPv6    | 337    |                | 0x00000000000001 |                       | 1         | Multicast Listener Report Message v2                                 |
| 2473 | 22.002217328 | 0.0.0.0           | 255.255.255.255   | DHCP      | 412    |                | 0x00000000000002 |                       | 0         | DHCP Request - Transaction ID 0xafc650cc                             |
| 2477 | 22.018768325 | 0.0.0.0           | 255.255.255.255   | DHCP      | 383    |                | 0x00000000000002 |                       | 1         | DHCP Request - Transaction ID 0xafc650cc                             |
| 2479 | 22.032729371 | Shenzhen_2f:f6:f0 | Broadcast         | 802.11    | 152    |                |                  |                       |           | Beacon frame, SN=0, FN=0, Flags=....., BI=100, SSID="testnetwork"    |
| 2480 | 22.065594241 | 22:61:71:23:de:e6 | Broadcast         | ARP       | 118    |                | 0x00000000000003 |                       | 0         | Who has 192.168.100.254? Tell 192.168.100.2                          |
| 2482 | 22.066630889 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 162    | 3              | 0x00000000000002 |                       | 0         | Key (Message 1 of 4)                                                 |
| 2484 | 22.067537712 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL     | 218    | 4              | 0x00000000000003 |                       | 0         | Key (Message 3 of 4)                                                 |
| 2485 | 22.070272175 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL     | 211    | 3              | 0x00000000000004 |                       | 0         | Key (Message 2 of 4)                                                 |
| 2486 | 22.071597717 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL     | 66     |                |                  |                       |           | Disassociate, SN=2775, FN=0, Flags=.....C                            |
| 2488 | 22.072480804 | 22:61:71:23:de:e6 | Broadcast         | ARP       | 89     |                | 0x00000000000001 |                       | 1         | Who has 192.168.100.254? Tell 192.168.100.2                          |

Figure 21: *tpk-rand* variant of KRACK and Disassociation as a response by device no. 12 (Test 4)

As seen in Figure 21 (see highlighted frame), the `--tpk-rand` test would make some devices send a Disassociate (management) frame with an “Unspecified reason” code after having received EAPOL Message 1 and EAPOL Message 3 in a row. This also occurred for some devices with the `-tpk` and the `--gtkinit` options (Tests 3 and 5). It seems that the unexpected transmission of the EAPOL messages makes the client disconnect from the WLAN. Further in the communication, the device then reconnects to the network going through a new 4-way handshake. For the Sony Xperia phone (device no. 16), this “disconnect cycle” behaviour was repeated throughout most of the tests.

Test 5 is described as being rather “unreliable” by Vanhoef in [27], “*because any missed handshake messages cause synchronization issues*”. It tests if the device installs the given RSC value in the 4-way handshake. When inspecting Wireshark captures of this test, an anomaly was observed: the RSC value set in the EAPOL Message 3 by *hostapd* should have been set to a higher value. In practice, it was observed that this wasn’t the case: the provided RSC was always a zero-byte value for all EAPOL messages (see Figure 22).

| No. | Time        | Source            | Destination       | Protocol | Length | Replay Counter | WPA Key RSC      | COMP Ext.        | Initialization Vector | Key Index | Info                 |
|-----|-------------|-------------------|-------------------|----------|--------|----------------|------------------|------------------|-----------------------|-----------|----------------------|
| 35  | 0.767276007 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 146    | 1              | 0000000000000000 |                  |                       |           | Key (Message 1 of 4) |
| 37  | 0.777418288 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 195    | 1              | 0000000000000000 |                  |                       |           | Key (Message 2 of 4) |
| 38  | 0.782320130 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 202    | 2              | 0000000000000000 |                  |                       |           | Key (Message 3 of 4) |
| 40  | 0.783330036 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 2              | 0000000000000000 |                  |                       |           | Key (Message 4 of 4) |
| 181 | 3.130389957 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 162    | 3              | 0000000000000000 | 0x0000000000000B |                       | 0         | Key (Message 1 of 4) |
| 182 | 3.134017698 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 211    | 3              | 0000000000000000 | 0x00000000000014 |                       | 0         | Key (Message 2 of 4) |
| 183 | 3.139006406 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 4              | 0000000000000000 | 0x0000000000000C |                       | 0         | Key (Message 3 of 4) |
| 185 | 3.140808635 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 4              | 0000000000000000 | 0x00000000000015 |                       | 0         | Key (Message 4 of 4) |
| 251 | 5.179660947 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 162    | 5              | 0000000000000000 | 0x00000000000001 |                       | 0         | Key (Message 1 of 4) |
| 252 | 5.184070587 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 211    | 5              | 0000000000000000 | 0x00000000000002 |                       | 0         | Key (Message 2 of 4) |
| 254 | 5.187742462 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 6              | 0000000000000000 | 0x00000000000002 |                       | 0         | Key (Message 3 of 4) |
| 255 | 5.190569508 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 6              | 0000000000000000 | 0x00000000000003 |                       | 0         | Key (Message 4 of 4) |
| 347 | 7.226459699 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 162    | 7              | 0000000000000000 | 0x00000000000007 |                       | 0         | Key (Message 1 of 4) |
| 348 | 7.234175804 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 211    | 7              | 0000000000000000 | 0x00000000000009 |                       | 0         | Key (Message 2 of 4) |
| 350 | 7.238841548 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 8              | 0000000000000000 | 0x00000000000008 |                       | 0         | Key (Message 3 of 4) |
| 351 | 7.243278240 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 8              | 0000000000000000 | 0x0000000000000A |                       | 0         | Key (Message 4 of 4) |
| 512 | 9.280092882 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 162    | 9              | 0000000000000000 | 0x00000000000001 |                       | 0         | Key (Message 1 of 4) |
| 513 | 9.283649744 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 211    | 9              | 0000000000000000 | 0x00000000000001 |                       | 0         | Key (Message 2 of 4) |
| 515 | 9.288478611 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 10             | 0000000000000000 | 0x00000000000002 |                       | 0         | Key (Message 3 of 4) |
| 516 | 9.291250448 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 10             | 0000000000000000 | 0x00000000000002 |                       | 0         | Key (Message 4 of 4) |

Figure 22: Wrong behaviour by *hostapd* for Test 5 (device no. 12)

Debugging this issue wasn’t clear until the code was reviewed again: a logic bug seems to reside at line 541 of the `run()` method. Indeed, the previous conditional statement (line

533) explicitly excludes the case where the “gtkinit” option is on. Therefore, this option can never be acknowledged to set a higher RSC value when sending another EAPOL Message 3, as shown in Figure 23. An issue was raised on the GitHub repository to address this.

```

528 # 1. Test the 4-way handshake
529 if self.options.variant == TestOptions.Fourway and self.options.gtkinit and client.vuln_bcast != ClientState.VULNERABLE:
530     # Execute a new handshake to test stations that don't accept a retransmitted message 3
531     hostapd_command(self.hostapd_ctrl, "RENEW_PTK " + client.mac)
532     # T000: wait untill 4-way handshake completed? And detect failures (it's sensitive to frame losses)?
533     elif self.options.variant == TestOptions.Fourway and not self.options.gtkinit and client.vuln_4way != ClientState.VULNERABLE:
534         # First inject a message 1 if requested using the TPTK option
535         if self.options.tptk == TestOptions.TptkReplay:
536             hostapd_command(self.hostapd_ctrl, "RESEND_M1 " + client.mac)
537         elif self.options.tptk == TestOptions.TptkRand:
538             hostapd_command(self.hostapd_ctrl, "RESEND_M1 " + client.mac + " change-anonce")
539
540     # Note that we rely on an encrypted message 4 as reply to detect pairwise key reinstallations reinstallations.
541     hostapd_command(self.hostapd_ctrl, "RESEND_M3 " + client.mac + " maxrsc" if self.options.gtkinit else "")
542
543 # 2. Test if broadcast ARP request are accepted by the client. Keep injecting even

```

Figure 23: Broken logic in the run () method

Considering this script malfunction, the code was manually corrected to allow the “gtkinit” option to work as expected during the GTK reinstallation test of a 4-way handshake. For unknown reasons, recapturing the traffic generated by Test 5 on the Fairphone gave a similar output: it seems that the 4-way handshake is performed (as expected with the “gtkinit” option), but retransmitting EAPOL Message 3 is on the other hand rarely performed. As the origin of this issue remains unidentified, the “gtkinit” test (Test 5) is set aside. For some devices, the test started the “disconnect cycle” mentioned previously.

Moving to Test 6, only 3 devices were detected as being vulnerable to GTK reinstallation in the group key handshake: the Samsung S5 (device no. 4), one iPad 2 (device no. 22) and the iPhone 5 (device no. 23). Most of the devices were not vulnerable to this test.

Finally, 5 devices were declared test-positive to Test 7, meaning that they reinstalled the RSC value during the group key handshake to 0. This test correctly assigned a high RSC value in EAPOL group message 1, to the contrary of Test 5. Given the previous issue with the “gtkinit” option, the test is briefly investigated here for the Fairphone FP4 (Figure 24).

| No. | Time         | Source            | Destination       | Protocol | Length | Replay Counter | WPA Key RSC      | CCMP Ext. Initialization Vector | Key Index | Info                       |
|-----|--------------|-------------------|-------------------|----------|--------|----------------|------------------|---------------------------------|-----------|----------------------------|
| 131 | 7.524702920  | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 146    | 1              | 0000000000000000 |                                 |           | Key (Message 1 of 4)       |
| 133 | 7.532212615  | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 195    | 1              | 0000000000000000 |                                 |           | Key (Message 2 of 4)       |
| 136 | 7.541924498  | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 202    | 2              | 0000000000000000 |                                 |           | Key (Message 3 of 4)       |
| 137 | 7.543162147  | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 2              | 0000000000000000 |                                 |           | Key (Message 4 of 4)       |
| 248 | 10.320828636 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 194    | 3              | ffff020000000000 | 0x0000000000000008              | 0         | Key (Group Message 1 of 2) |
| 249 | 10.330759243 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 3              | 0000000000000000 | 0x0000000000000015              | 0         | Key (Group Message 2 of 2) |
| 284 | 12.477016577 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 194    | 4              | ffff020000000000 | 0x000000000000000C              | 0         | Key (Group Message 1 of 2) |
| 285 | 12.485417109 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 4              | 0000000000000000 | 0x0000000000000016              | 0         | Key (Group Message 2 of 2) |
| 349 | 14.732463082 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 194    | 5              | ffff020000000000 | 0x0000000000000013              | 0         | Key (Group Message 1 of 2) |
| 350 | 14.739092551 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 5              | 0000000000000000 | 0x000000000000001F              | 0         | Key (Group Message 2 of 2) |
| 373 | 16.588089481 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 194    | 6              | ffff020000000000 | 0x0000000000000014              | 0         | Key (Group Message 1 of 2) |
| 375 | 16.594478008 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 6              | 0000000000000000 | 0x0000000000000020              | 0         | Key (Group Message 2 of 2) |
| 409 | 18.644376115 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 194    | 7              | ffff020000000000 | 0x0000000000000017              | 0         | Key (Group Message 1 of 2) |
| 411 | 18.657146069 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 7              | 0000000000000000 | 0x0000000000000023              | 0         | Key (Group Message 2 of 2) |
| 467 | 20.672278197 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 194    | 8              | ffff020000000000 | 0x000000000000001C              | 0         | Key (Group Message 1 of 2) |
| 469 | 20.680619507 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 8              | 0000000000000000 | 0x0000000000000029              | 0         | Key (Group Message 2 of 2) |
| 522 | 22.588352332 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 194    | 9              | ffff020000000000 | 0x0000000000000020              | 0         | Key (Group Message 1 of 2) |
| 524 | 22.595666386 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 9              | 0000000000000000 | 0x000000000000002F              | 0         | Key (Group Message 2 of 2) |
| 563 | 24.872206365 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 194    | 10             | ffff020000000000 | 0x0000000000000023              | 0         | Key (Group Message 1 of 2) |
| 564 | 24.880649457 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 10             | 0000000000000000 | 0x0000000000000033              | 0         | Key (Group Message 2 of 2) |
| 589 | 26.836414507 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 194    | 11             | ffff020000000000 | 0x0000000000000024              | 0         | Key (Group Message 1 of 2) |
| 590 | 26.846327963 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 11             | 0000000000000000 | 0x0000000000000034              | 0         | Key (Group Message 2 of 2) |

Figure 24: RSC value in EAPOL group message 1 is not set by the client (device no. 12, Test 7)

The command line output states that the device installs the provided RSC in the group key handshake (which would be good). However, this appeared to be incorrect. The AP sends EAPOL group message 1 to the Fairphone with a high RSC value. As for the PN bytes in the CCMP Header, the value is read in reverse order, meaning that the RSC value sent is “00 00 00 00 00 02 FF FF” (in hexadecimal, or 196’607 in decimal). Instead of responding with an incremented RSC value, the client responds each time with an all-zero RSC.

## 8.2 Case Studies of KRACK Tests

Going further in the discussion of results, this subchapter will present a selection of cases where the traffic was captured. First, the example of a device protected against the main KRACK attack (Test 1) is presented. Then, various examples of test-positives results are discussed.

### 8.2.1 KRACK against Fairphone FP4 (Android v12)

The case of the Fairphone FP4 (device no. 12) is described first. In [Figure 25](#), the EAPOL Messages exchanged during the performance of Test 1 against the FP4 are displayed.

| No.  | Time         | Source            | Destination       | Protocol | Length | Replay Counter | CCMP Ext. Initialization Vector | Key Index | Info                 |
|------|--------------|-------------------|-------------------|----------|--------|----------------|---------------------------------|-----------|----------------------|
| 159  | 3.014150017  | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 146    | 1              |                                 |           | Key (Message 1 of 4) |
| 161  | 3.026031524  | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 195    | 1              |                                 |           | Key (Message 2 of 4) |
| 163  | 3.029571288  | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 202    | 2              |                                 |           | Key (Message 3 of 4) |
| 164  | 3.035435538  | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 2              |                                 |           | Key (Message 4 of 4) |
| 207  | 3.340196607  | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 3              | 0x00000000000008                | 0         | Key (Message 3 of 4) |
| 208  | 3.342304719  | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 3              |                                 |           | Key (Message 4 of 4) |
| 349  | 5.493430471  | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 4              | 0x00000000000008                | 0         | Key (Message 3 of 4) |
| 350  | 5.497000638  | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 4              |                                 |           | Key (Message 4 of 4) |
| 461  | 7.538878103  | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 5              | 0x0000000000000C                | 0         | Key (Message 3 of 4) |
| 463  | 7.540399053  | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 5              |                                 |           | Key (Message 4 of 4) |
| 593  | 9.687527594  | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 6              | 0x00000000000013                | 0         | Key (Message 3 of 4) |
| 594  | 9.689721297  | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 6              |                                 |           | Key (Message 4 of 4) |
| 685  | 11.839246902 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 7              | 0x00000000000014                | 0         | Key (Message 3 of 4) |
| 686  | 11.840721990 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 7              |                                 |           | Key (Message 4 of 4) |
| 769  | 13.580032361 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 8              |                                 |           | Key (Message 4 of 4) |
| 775  | 13.614938092 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 8              | 0x00000000000016                | 0         | Key (Message 3 of 4) |
| 909  | 15.730471056 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 9              | 0x0000000000001C                | 0         | Key (Message 3 of 4) |
| 910  | 15.735093991 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 9              |                                 |           | Key (Message 4 of 4) |
| 1019 | 17.665420497 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 10             | 0x0000000000001D                | 0         | Key (Message 3 of 4) |
| 1022 | 17.669600713 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 10             |                                 |           | Key (Message 4 of 4) |
| 1136 | 19.716928098 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 11             | 0x00000000000021                | 0         | Key (Message 3 of 4) |
| 1139 | 19.722463987 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 11             |                                 |           | Key (Message 4 of 4) |
| 1255 | 21.972730718 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 12             | 0x00000000000024                | 0         | Key (Message 3 of 4) |
| 1264 | 22.126374532 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 12             |                                 |           | Key (Message 4 of 4) |
| 1363 | 24.018772820 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 13             | 0x00000000000025                | 0         | Key (Message 3 of 4) |
| 1364 | 24.024758215 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 13             |                                 |           | Key (Message 4 of 4) |
| 1449 | 25.958719836 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 14             | 0x00000000000026                | 0         | Key (Message 3 of 4) |
| 1450 | 25.962006816 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 14             |                                 |           | Key (Message 4 of 4) |
| 1616 | 28.006457132 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 15             | 0x0000000000002D                | 0         | Key (Message 3 of 4) |
| 1618 | 28.010056429 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 15             |                                 |           | Key (Message 4 of 4) |
| 1719 | 30.257419744 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | EAPOL    | 218    | 16             | 0x0000000000002E                | 0         | Key (Message 3 of 4) |
| 1720 | 30.260074371 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 16             |                                 |           | Key (Message 4 of 4) |

Figure 25: Test 1 on device no. 12 (Fairphone FP4) with the first KRACK attempt highlighted

It shows the multiple EAPOL Message 3s which are sent again and again by the AP, and the corresponding EAPOL Message 4s sent by the Fairphone in response. The first retransmission of an EAPOL Message 3 takes place on the (highlighted) 207<sup>th</sup> frame, meaning that it is the first occurrence of a key reinstallation test. Although the EAPOL Message 3s of the AP are always encrypted using the TK (see “Key Index” column), none of the EAPOL Message 4s are encrypted by the Fairphone. It can be now checked if encrypted messages sent by the client (starting from the 208<sup>th</sup> frame) reused PTK-IVs, which would indicate a PTK



reinstallation occurred. In Figure 26, the PTK-IV incrementation is correctly done by the Fairphone. It is not true for frames sent by the AP though, as replayed broadcast ARPs are sent on purpose to test for GTK reinstallation for example.

| No. | Time         | Source                    | Destination       | Protocol | Length | Replay Counter | CCMP Ext. Initialization Vector | Key Index | Info                                   |
|-----|--------------|---------------------------|-------------------|----------|--------|----------------|---------------------------------|-----------|----------------------------------------|
| 168 | 3.093198093  | ::                        | ff02::16          | ICMPv6   | 286    |                | 0x000000000001                  | 0         | Multicast Listener Report Message v2   |
| 171 | 3.133263841  | 0.0.0.0                   | 255.255.255.255   | DHCP     | 412    |                | 0x000000000002                  | 0         | DHCP Request - Transaction ID 0x149ea  |
| 178 | 3.203917609  | 22:61:71:23:de:e6         | Broadcast         | ARP      | 118    |                | 0x000000000003                  | 0         | Who has 192.168.100.254? Tell 192.168. |
| 183 | 3.213548881  | 192.168.100.2             | 192.168.100.254   | TCP      | 150    |                | 0x000000000004                  | 0         | 45938 → 853 [SYN] Seq=0 Win=65535 Len= |
| 190 | 3.277561222  | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |                | 0x000000000005                  | 0         | Standard query 0x4f75 A connectivitych |
| 193 | 3.284015645  | 192.168.100.2             | 192.168.100.254   | DNS      | 150    |                | 0x000000000006                  | 0         | Standard query 0x6322 A www.google.com |
| 197 | 3.290029171  | 192.168.100.2             | 192.168.100.254   | DNS      | 150    |                | 0x000000000007                  | 0         | Standard query 0x6322 A www.google.com |
| 198 | 3.290050865  | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |                | 0x000000000008                  | 0         | Standard query 0x4f75 A connectivitych |
| 209 | 3.356077099  | 192.168.100.2             | 192.168.100.254   | DNS      | 150    |                | 0x000000000009                  | 0         | Standard query 0xe61b A www.google.com |
| 210 | 3.356959171  | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |                | 0x00000000000a                  | 0         | Standard query 0x184c A connectivitych |
| 213 | 3.361770224  | 192.168.100.2             | 192.168.100.254   | DNS      | 150    |                | 0x00000000000b                  | 0         | Standard query 0xe61b A www.google.com |
| 217 | 3.374008165  | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |                | 0x00000000000c                  | 0         | Standard query 0x184c A connectivitych |
| 249 | 4.001756090  | ::                        | ff02::1:ff23:dee6 | ICMPv6   | 162    |                | 0x00000000000d                  | 0         | Neighbor Solicitation for fe80::2061:7 |
| 261 | 4.120582984  | ::                        | ff02::16          | ICMPv6   | 286    |                | 0x00000000000e                  | 0         | Multicast Listener Report Message v2   |
| 293 | 4.772819462  | fe80::2061:71ff:fe23:dee6 | ff02::16          | ICMPv6   | 226    |                | 0x00000000000f                  | 0         | Multicast Listener Report Message v2   |
| 294 | 4.772895479  | fe80::2061:71ff:fe23:dee6 | ff02::2           | ICMPv6   | 146    |                | 0x000000000010                  | 0         | Router Solicitation from 22:61:71:23:d |
| 295 | 4.772897535  | fe80::2061:71ff:fe23:dee6 | ff02::16          | ICMPv6   | 166    |                | 0x000000000011                  | 0         | Multicast Listener Report Message v2   |
| 313 | 5.01737667   | fe80::2061:71ff:fe23:dee6 | ff02::16          | ICMPv6   | 226    |                | 0x000000000012                  | 0         | Multicast Listener Report Message v2   |
| 322 | 5.151266682  | fe80::2061:71ff:fe23:dee6 | ff02::16          | ICMPv6   | 166    |                | 0x000000000013                  | 0         | Multicast Listener Report Message v2   |
| 515 | 8.474843986  | 192.168.100.2             | 192.168.100.254   | DNS      | 150    |                | 0x000000000014                  | 0         | Standard query 0x4596 A www.google.com |
| 516 | 8.474879449  | 192.168.100.2             | 192.168.100.254   | DNS      | 166    |                | 0x000000000015                  | 0         | Standard query 0x628e A www.google.com |
| 517 | 8.474885944  | 192.168.100.2             | 192.168.100.254   | DNS      | 181    |                | 0x000000000016                  | 0         | Standard query 0x7830 A connectivitych |
| 521 | 8.478801626  | 22:61:71:23:de:e6         | Shenzhen_2f:f6:f0 | ARP      | 118    |                | 0x000000000017                  | 0         | 192.168.100.2 is at 22:61:71:23:de:e6  |
| 527 | 8.500429048  | 192.168.100.2             | 192.168.100.254   | DNS      | 150    |                | 0x000000000018                  | 0         | Standard query 0x4596 A www.google.com |
| 531 | 8.502937625  | 192.168.100.2             | 192.168.100.254   | DNS      | 181    |                | 0x000000000019                  | 0         | Standard query 0x7830 A connectivitych |
| 532 | 8.503829239  | 192.168.100.2             | 192.168.100.254   | DNS      | 166    |                | 0x00000000001a                  | 0         | Standard query 0x628e A www.google.com |
| 546 | 8.671082042  | fe80::2061:71ff:fe23:dee6 | ff02::2           | ICMPv6   | 146    |                | 0x00000000001b                  | 0         | Router Solicitation from 22:61:71:23:d |
| 758 | 13.520699714 | 192.168.100.2             | 192.168.100.254   | DNS      | 166    |                | 0x00000000001c                  | 0         | Standard query 0x11a6 A www.google.com |
| 765 | 13.563478353 | 192.168.100.2             | 192.168.100.254   | DNS      | 166    |                | 0x00000000001d                  | 0         | Standard query 0x11a6 A www.google.com |
| 823 | 14.640462103 | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |                | 0x00000000001e                  | 0         | Standard query 0x1974 A connectivitych |
| 824 | 14.640533088 | 192.168.100.2             | 192.168.100.254   | DNS      | 150    |                | 0x00000000001f                  | 0         | Standard query 0x6e34 A www.google.com |
| 829 | 14.648569074 | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |                | 0x000000000020                  | 0         | Standard query 0x1974 A connectivitych |
| 830 | 14.650211841 | 192.168.100.2             | 192.168.100.254   | DNS      | 150    |                | 0x000000000021                  | 0         | Standard query 0x6e34 A www.google.com |
| 836 | 14.665811306 | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |                | 0x000000000022                  | 0         | Standard query 0xf10f A connectivitych |

Figure 26: No PTK reinstallation, as no PTK-IV was reused on device no. 12 (with Test 1)

Note that for GTK reinstallation, it can be deduced from the presence (or absence) of replies to the replayed ARP requests. To confirm that the GTK didn't get reinstalled, the replayed ARP frames must be investigated (see Figure 27).

| No.  | Time         | Source            | Destination       | Protocol | Length | Replay Counter | CCMP Ext. Initialization Vector | Key Index | Info                                        |
|------|--------------|-------------------|-------------------|----------|--------|----------------|---------------------------------|-----------|---------------------------------------------|
| 178  | 3.203917609  | 22:61:71:23:de:e6 | Broadcast         | ARP      | 118    |                | 0x000000000003                  | 0         | Who has 192.168.100.254? Tell 192.168.100.2 |
| 179  | 3.208293934  | 22:61:71:23:de:e6 | Broadcast         | ARP      | 89     |                | 0x000000000003                  | 1         | Who has 192.168.100.254? Tell 192.168.100.2 |
| 182  | 3.211073781  | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | ARP      | 91     |                | 0x000000000002                  | 0         | 192.168.100.254 is at c0:1c:30:2f:f6:f0     |
| 287  | 4.624778591  | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000003                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 424  | 6.672283369  | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 520  | 8.476783528  | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | ARP      | 91     |                | 0x00000000000d                  | 0         | Who has 192.168.100.2? Tell 192.168.100.254 |
| 521  | 8.478801626  | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | ARP      | 118    |                | 0x000000000017                  | 0         | 192.168.100.2 is at 22:61:71:23:de:e6       |
| 539  | 8.512653339  | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 639  | 10.765822847 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 721  | 12.006776160 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 848  | 14.850869746 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 984  | 16.906628409 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000002                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 1183 | 18.953095900 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 1207 | 21.001858470 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 1311 | 22.842195147 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 1402 | 24.887974731 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 1528 | 26.832529998 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 1662 | 28.981190350 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 1793 | 30.930725160 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000002                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 1873 | 31.878888141 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | ARP      | 118    |                | 0x00000000003b                  | 0         | Who has 192.168.100.254? Tell 192.168.100.2 |
| 1874 | 31.881788546 | Shenzhen_2f:f6:f0 | 22:61:71:23:de:e6 | ARP      | 91     |                | 0x000000000034                  | 0         | 192.168.100.254 is at c0:1c:30:2f:f6:f0     |
| 1947 | 33.077236574 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 2042 | 35.125380700 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000002                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 2166 | 37.171321836 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 2251 | 39.219270330 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 2300 | 41.265194946 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 2483 | 43.310645994 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 2581 | 45.358026023 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 2720 | 47.405255535 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 2856 | 49.453818565 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |
| 2967 | 51.340958703 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | ARP      | 91     |                | 0x00000000004d                  | 0         | Who has 192.168.100.2? Tell 192.168.100.254 |
| 2969 | 51.342743307 | 22:61:71:23:de:e6 | Shenzhen_2f:f6:f0 | ARP      | 118    |                | 0x00000000004d                  | 0         | 192.168.100.2 is at 22:61:71:23:de:e6       |
| 2980 | 51.498686948 | Shenzhen_2f:f6:f0 | Broadcast         | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1   |

Figure 27: No replies from device no. 12 to replayed ARP requests (Test 1)

When inspecting these, only non-replayed ARP requests were replied to by the client. Such a response can be found for example in frames no. 520-521 or frames no. 2967-2969. May replayed broadcasted ARP requests have the GTK-IV value of “1” here (see IV values for frames with “1” in the “Key Index” column).

## 8.2.2 KRACK against Blackberry Bold 9700 (BlackberryOS v6)

Looking at a selection of positive tests, the BlackBerry (device no. 3) is the first one presented. Both the command line and the Wireshark capture confirm that a PTK-IV was reused. The script's output with debug option on is shown, stating that it is the PTK-IV value of 7 which was first reused (Figure 28). A Wireshark capture was also recorded during Test 1 (see Figure 29) for this device, showing that the IV value of 7 was the first reused nonce. Although it is not clear why precedent KRACK attempts succeeded in making the client reinstall the PTK, later attempts showed that other PTK-IVs were reused (such as the IV value of 8, in frames no. 9331 and 9859).

```
[11:50:34] Reset PN for GTK
[11:50:34] F4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:50:34] F4:0b:93:66:fc:9f: received a new message 4
[11:50:35] F4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 2 ARPs this interval)
[11:50:36] Reset PN for GTK
[11:50:36] F4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:50:37] F4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 3 ARPs this interval)
[11:50:37] F4:0b:93:66:fc:9f: transmitted data using IV=6 (seq=9)
[11:50:38] Reset PN for GTK
[11:50:38] F4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:50:38] F4:0b:93:66:fc:9f: received a new message 4
[11:50:39] F4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 4 ARPs this interval)
[11:50:39] F4:0b:93:66:fc:9f: transmitted data using IV=7 (seq=10)
[11:50:39] F4:0b:93:66:fc:9f: no pairwise IV resets seem to have occurred for one interval
[11:50:40] Reset PN for GTK
[11:50:40] F4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:50:40] F4:0b:93:66:fc:9f: didn't get reply received to broadcast ARPs during this interval
[11:50:41] F4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 1 ARPs this interval)
[11:50:41] F4:0b:93:66:fc:9f: transmitted data using IV=7 (seq=11)
[11:50:41] F4:0b:93:66:fc:9f: IV reuse detected (IV=7, seq=11). Client reinstalls the pairwise key in the 4-way handshake (this is bad)
[11:50:42] Reset PN for GTK
[11:50:42] F4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:50:42] F4:0b:93:66:fc:9f: received a new message 4
[11:50:43] F4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 2 ARPs this interval)
[11:50:43] F4:0b:93:66:fc:9f: transmitted data using IV=8 (seq=12)
[11:50:44] Reset PN for GTK
```

Figure 28: IV reuse statement on the command line by device no. 3 (Test 1)

| No.   | Time         | Source            | Destination       | Protocol | Length | Replay Counter | CCMP Ext. Initialization Vector | Key Index | Info                 |
|-------|--------------|-------------------|-------------------|----------|--------|----------------|---------------------------------|-----------|----------------------|
| 7446  | 23.369978388 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 10             | 0x0000000000006                 | 0         | Key (Message 4 of 4) |
| 7678  | 25.206383912 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 11             | 0x0000000000008                 | 0         | Key (Message 3 of 4) |
| 7680  | 25.213167726 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 11             | 0x0000000000007                 | 0         | Key (Message 4 of 4) |
| 8300  | 27.251173727 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 12             | 0x000000000000C                 | 0         | Key (Message 3 of 4) |
| 8302  | 27.259727681 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 12             | 0x0000000000007                 | 0         | Key (Message 4 of 4) |
| 9327  | 29.299047436 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 13             | 0x000000000000D                 | 0         | Key (Message 3 of 4) |
| 9331  | 29.304962540 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 13             | 0x0000000000008                 | 0         | Key (Message 4 of 4) |
| 9857  | 31.350899295 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 14             | 0x000000000000E                 | 0         | Key (Message 3 of 4) |
| 9859  | 31.359090495 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 14             | 0x0000000000008                 | 0         | Key (Message 4 of 4) |
| 10517 | 33.395927777 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 15             | 0x000000000000F                 | 0         | Key (Message 3 of 4) |
| 10519 | 33.403317403 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 15             | 0x0000000000009                 | 0         | Key (Message 4 of 4) |

Figure 29: IV reuse by device no. 3 (Test 1)

Note that according to the 802.11i standard, even a valid replayed encrypted frame should increment its packet number (i.e., IV). Indeed, “*The decryption processing prevents replay of MPDUs by validating that the PN in the MPDU is greater than the replay counter maintained for the session.*” [13, p. 184], with the MPDU standing for the MAC Protocol Data Unit, which can be simplified as the frame here.

Continuing the investigation, the GTK was not reinstalled according to the script (Figure 30, p. 63). In fact, the BlackBerry device never replied to any ARP request according to the Wireshark capture (Figure 31, p. 63).

```

[11:51:02] Reset PN for GTK
[11:51:02] f4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:51:02] f4:0b:93:66:fc:9f: received a new message 4
[11:51:03] f4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 4 ARPs this interval)
[11:51:03] f4:0b:93:66:fc:9f: transmitted data using IV=13 (seq=22)
[11:51:04] Reset PN for GTK
[11:51:04] f4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:51:04] f4:0b:93:66:fc:9f: didn't get reply received to broadcast ARPs during this interval
[11:51:04] f4:0b:93:66:fc:9f: Client DOESN'T reinstall the group key in the 4-way handshake (this is good)
[11:51:05] f4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 1 ARPs this interval)
[11:51:05] f4:0b:93:66:fc:9f: transmitted data using IV=13 (seq=23)
[11:51:06] Reset PN for GTK
[11:51:06] f4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:51:06] f4:0b:93:66:fc:9f: received a new message 4
[11:51:07] f4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 2 ARPs this interval)
[11:51:07] f4:0b:93:66:fc:9f: transmitted data using IV=14 (seq=24)
[11:51:08] Reset PN for GTK
[11:51:08] f4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:51:09] f4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 3 ARPs this interval)
[11:51:09] f4:0b:93:66:fc:9f: transmitted data using IV=14 (seq=25)
[11:51:10] Reset PN for GTK

```

Figure 30: No GTK reinstallation in the 4-way handshake for device no. 3 (Test 1)

| No.   | Time         | Source            | Destination | Protocol | Length | Replay Counter | CCMP Ext. Initialization Vector | Key Index | Info                                      |
|-------|--------------|-------------------|-------------|----------|--------|----------------|---------------------------------|-----------|-------------------------------------------|
| 2870  | 9.914392222  | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000004                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 3941  | 11.958340770 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 4269  | 14.005712441 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 4991  | 16.264124427 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 6049  | 18.304607567 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 6299  | 20.351818797 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 7315  | 22.194452704 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 7542  | 24.242187800 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 8196  | 26.287633771 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 9211  | 28.333631112 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 9454  | 30.381711623 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 10395 | 32.430078822 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 10651 | 34.481655426 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 11374 | 36.530317401 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 12159 | 38.578495445 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 12406 | 40.627072372 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 13334 | 42.677344003 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 13565 | 44.724477819 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 14161 | 46.774313444 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 15086 | 48.823297131 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 15250 | 50.871043060 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |
| 16230 | 52.917941588 | Shenzhen_2f:f6:f0 | Broadcast   | ARP      | 89     |                | 0x000000000001                  | 1         | Who has 192.168.100.2? Tell 192.168.100.1 |

Figure 31: Absence of replies to replayed ARP requests for device no. 3 (Test 1)

Concerning PTK reinstallation, the same pattern occurred when performing Test 3 (TPTK variant) on the device, as can be seen in [Figure 32](#) and [Figure 33](#) (p. 64). PTK-IVs reuse was observed, but not GTK reinstallation.

```

(worv)~(root@kali):~/home/kali/krackattacks-scripts/krackattack
# python krack-test-client.py --gtk --debug
[11:59:13] Note: disable Wi-Fi in network manager & disable hardware encryption. Both may interfere with this script.
[11:59:15] Starting hostapd ...
Configuration file: /home/kali/krackattacks-scripts/krackattack/hostapd.conf
Using interface wlan0 with hwaddr c0:1c:30:2f:f6:f0 and ssid "testnetwork"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
[11:59:16] Ready. Connect to this Access Point to start the tests. Make sure the client requests an IP using DHCP!
[11:59:17] Reset PN for GTK
[11:59:19] Reset PN for GTK
[11:59:21] Reset PN for GTK
[11:59:23] Reset PN for GTK
[11:59:25] Reset PN for GTK
wlan0: STA f4:0b:93:66:fc:9f IEEE 802.11: authenticated
wlan0: STA f4:0b:93:66:fc:9f IEEE 802.11: associated (aid 1)
wlan0: AP-STA-CONNECTED f4:0b:93:66:fc:9f
wlan0: STA f4:0b:93:66:fc:9f RADIUS: starting accounting session B16C4E915E2E015D
[11:59:26] f4:0b:93:66:fc:9f: 4-way handshake completed (RSN)
[11:59:26] f4:0b:93:66:fc:9f: DHCP reply 192.168.100.2 to f4:0b:93:66:fc:9f
[11:59:26] f4:0b:93:66:fc:9f: DHCP reply 192.168.100.2 to f4:0b:93:66:fc:9f
[11:59:26] f4:0b:93:66:fc:9f: Removing client from DHCP leases
[11:59:26] f4:0b:93:66:fc:9f: transmitted data using IV=1 (seq=1)
[11:59:27] f4:0b:93:66:fc:9f: transmitted data using IV=2 (seq=2)
[11:59:27] Reset PN for GTK
[11:59:27] f4:0b:93:66:fc:9f: sending a new 4-way message 1 (with same ANonce)
[11:59:27] f4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:59:27] f4:0b:93:66:fc:9f: received a new message 2
[11:59:27] f4:0b:93:66:fc:9f: transmitted data using IV=3 (seq=3)
[11:59:27] f4:0b:93:66:fc:9f: received a new message 4
[11:59:28] f4:0b:93:66:fc:9f: transmitted data using IV=4 (seq=4)
[11:59:29] Reset PN for GTK
[11:59:29] f4:0b:93:66:fc:9f: sending a new 4-way message 1 (with same ANonce)
[11:59:29] f4:0b:93:66:fc:9f: sending a new 4-way message 3 where the GTK has a zero RSC
[11:59:29] f4:0b:93:66:fc:9f: received a new message 4
[11:59:29] f4:0b:93:66:fc:9f: transmitted data using IV=4 (seq=5)
[11:59:29] f4:0b:93:66:fc:9f: IV reuse detected (IV=4, seq=5). client reinstalls the pairwise key in the 4-way handshake (this is bad)
[11:59:29] f4:0b:93:66:fc:9f: transmitted data using IV=5 (seq=6)
[11:59:31] Reset PN for GTK

```

Figure 32: *tptk* variant of KRACK is successful on device no. 3 (Test 3)



| No.  | Time         | Source            | Destination       | Protocol | Length | Replay Counter | CCMP Ext. Initialization Vector | Key Index | Info                   |
|------|--------------|-------------------|-------------------|----------|--------|----------------|---------------------------------|-----------|------------------------|
| 2615 | 7.928459588  | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 146    | 1              |                                 |           | Key (Message 1 of 4)   |
| 2621 | 7.932641675  | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 195    | 1              |                                 |           | Key (Message 2 of 4)   |
| 2629 | 7.935830559  | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 202    | 2              |                                 |           | Key (Message 3 of 4)   |
| 2631 | 7.940800821  | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 2              |                                 |           | Key (Message 4 of 4)   |
| 2754 | 8.879987759  | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 162    | 3              | 0x0000000000003                 |           | 0 Key (Message 1 of 4) |
| 2755 | 8.882134374  | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 211    | 3              | 0x0000000000003                 |           | 0 Key (Message 2 of 4) |
| 2782 | 9.083708802  | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 4              | 0x0000000000004                 |           | 0 Key (Message 3 of 4) |
| 2784 | 9.089272957  | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 4              | 0x0000000000004                 |           | 0 Key (Message 4 of 4) |
| 3076 | 10.940314884 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 162    | 5              | 0x0000000000005                 |           | 0 Key (Message 1 of 4) |
| 3095 | 10.951181901 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 6              | 0x0000000000006                 |           | 0 Key (Message 3 of 4) |
| 3100 | 10.956282313 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 211    | 5              | 0x0000000000004                 |           | 0 Key (Message 2 of 4) |
| 3125 | 10.960002023 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 6              | 0x0000000000005                 |           | 0 Key (Message 4 of 4) |
| 3617 | 12.974113647 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 7              | 0x0000000000007                 |           | 0 Key (Message 3 of 4) |
| 3620 | 12.982301237 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 7              | 0x0000000000005                 |           | 0 Key (Message 4 of 4) |
| 4519 | 15.022018738 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 8              | 0x0000000000008                 |           | 0 Key (Message 3 of 4) |
| 4521 | 15.029075566 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 8              | 0x0000000000006                 |           | 0 Key (Message 4 of 4) |
| 4784 | 17.068264335 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 9              | 0x0000000000009                 |           | 0 Key (Message 3 of 4) |
| 4785 | 17.075149624 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 9              | 0x0000000000006                 |           | 0 Key (Message 4 of 4) |
| 5591 | 19.120656960 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 10             | 0x000000000000A                 |           | 0 Key (Message 3 of 4) |
| 5593 | 19.131553960 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 10             | 0x0000000000007                 |           | 0 Key (Message 4 of 4) |
| 5812 | 21.161040631 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 218    | 11             | 0x000000000000B                 |           | 0 Key (Message 3 of 4) |
| 5815 | 21.168508974 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 11             | 0x0000000000007                 |           | 0 Key (Message 4 of 4) |

Figure 33: Multiple IV reuse due to repeated PTK reinstallations on device no. 3 (Test 3)

The first *tpdk* variant attack worked (i.e., the successive transmission of EAPOL Message 1 and EAPOL Message 3) and lead to nonce reuse. The second and third highlighted pairs of frames represent a standard EAPOL Message 3 retransmission vulnerability. This occurred because the script already detected after the first *tpdk* test that the device was vulnerable, and thus didn't send EAPOL Message 1 anymore. This observation confirms the previous result of Test 1.

The last reviewed test for the Blackberry device is the “group-gtkinit” test (Test 7). On one hand, device no. 3 always sets its RSC value to 0 in EAPOL group message 2, although the AP sets it at a high value in EAPOL group message 1 (Figure 34). On the other hand, the command-line states that the Blackberry correctly resets the GTK counter (Figure 35, p. 65).

| No.  | Time         | Source            | Destination       | Protocol | Length | Replay Counter | WPA Key RSC      | CCMP Ext. Initialization Vector | Key Index | Info                         |
|------|--------------|-------------------|-------------------|----------|--------|----------------|------------------|---------------------------------|-----------|------------------------------|
| 2579 | 16.637741134 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 146    | 1              | 0000000000000000 |                                 |           | Key (Message 1 of 4)         |
| 2580 | 16.644253516 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 195    | 1              | 0000000000000000 |                                 |           | Key (Message 2 of 4)         |
| 2582 | 16.649112516 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 202    | 2              | 0000000000000000 |                                 |           | Key (Message 3 of 4)         |
| 2583 | 16.652442528 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 2              | 0000000000000000 |                                 |           | Key (Message 4 of 4)         |
| 2899 | 19.371192505 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 194    | 3              | ffff020000000000 | 0x0000000000003                 |           | 0 Key (Group Message 1 of 2) |
| 2903 | 19.376296647 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 3              | 0000000000000000 | 0x00000000000069                |           | 0 Key (Group Message 2 of 2) |
| 3153 | 21.423283636 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 194    | 4              | ffff020000000000 | 0x00000000000094                |           | 0 Key (Group Message 1 of 2) |
| 3154 | 21.429794287 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 4              | 0000000000000000 | 0x0000000000006A                |           | 0 Key (Group Message 2 of 2) |
| 3794 | 23.675124638 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 194    | 5              | ffff020000000000 | 0x00000000000005                |           | 0 Key (Group Message 1 of 2) |
| 3808 | 23.684388463 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 5              | 0000000000000000 | 0x0000000000006B                |           | 0 Key (Group Message 2 of 2) |
| 4185 | 25.513498779 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 194    | 6              | ffff020000000000 | 0x00000000000006                |           | 0 Key (Group Message 1 of 2) |
| 4187 | 25.519655821 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 6              | 0000000000000000 | 0x0000000000006C                |           | 0 Key (Group Message 2 of 2) |
| 4442 | 27.559953192 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 194    | 7              | ffff020000000000 | 0x00000000000007                |           | 0 Key (Group Message 1 of 2) |
| 4443 | 27.564418044 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 7              | 0000000000000000 | 0x0000000000006D                |           | 0 Key (Group Message 2 of 2) |
| 4691 | 29.685955801 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 194    | 8              | ffff020000000000 | 0x00000000000008                |           | 0 Key (Group Message 1 of 2) |
| 4695 | 29.611504967 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 8              | 0000000000000000 | 0x0000000000006E                |           | 0 Key (Group Message 2 of 2) |
| 4950 | 31.652778438 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 194    | 9              | ffff020000000000 | 0x00000000000009                |           | 0 Key (Group Message 1 of 2) |
| 4951 | 31.660816251 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 9              | 0000000000000000 | 0x0000000000006F                |           | 0 Key (Group Message 2 of 2) |
| 5194 | 33.702962632 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 194    | 10             | ffff020000000000 | 0x0000000000000A                |           | 0 Key (Group Message 1 of 2) |
| 5195 | 33.706401873 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 10             | 0000000000000000 | 0x00000000000070                |           | 0 Key (Group Message 2 of 2) |
| 5416 | 35.747694699 | Shenzhen_2f:f6:f0 | BlackBer_66:fc:9f | EAPOL    | 194    | 11             | ffff020000000000 | 0x0000000000000B                |           | 0 Key (Group Message 1 of 2) |
| 5418 | 35.752753203 | BlackBer_66:fc:9f | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 11             | 0000000000000000 | 0x00000000000071                |           | 0 Key (Group Message 2 of 2) |

Figure 34: As for the Fairphone, the RSC value is not incremented for the Blackberry (Test 7)

Given that the “gtkinit” option (Test 5) already produced unexpected behaviour, this additional observation for the “group-gtkinit” (Test 7) diminishes the confidence one can have in these 2 tests.

```

[12:15:12] Renewed the group key
[12:15:13] f4:0b:93:66:fc:9f: Send group message 1 for the same GTK and max RSC
[12:15:13] f4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 3 ARPs this interval)
[12:15:13] f4:0b:93:66:fc:9f: received a new group message 2
[12:15:13] f4:0b:93:66:fc:9f: transmitted data using IV=124 (seq=221)
[12:15:14] Renewed the group key
[12:15:15] f4:0b:93:66:fc:9f: Send group message 1 for the same GTK and max RSC
[12:15:15] f4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 4 ARPs this interval)
[12:15:15] f4:0b:93:66:fc:9f: received a new group message 2
[12:15:15] f4:0b:93:66:fc:9f: transmitted data using IV=125 (seq=222)
[12:15:16] Renewed the group key
[12:15:16] f4:0b:93:66:fc:9f: didn't get reply received to broadcast ARPs during this interval
[12:15:16] f4:0b:93:66:fc:9f: Client installs the group key in the group key handshake with the given replay counter (this is good)
[12:15:17] f4:0b:93:66:fc:9f: Send group message 1 for the same GTK and max RSC
[12:15:17] f4:0b:93:66:fc:9f: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 1 ARPs this interval)
[12:15:17] f4:0b:93:66:fc:9f: received a new group message 2
[12:15:17] f4:0b:93:66:fc:9f: transmitted data using IV=126 (seq=223)
[12:15:18] Renewed the group key

```

Figure 35: Blackberry device allegedly uses the given RSC value according to Test 7 (device no. 3)

To compare previous results of the Fairphone and the Blackberry devices for the “group-gtkinit” test, 2 other devices, which were detected as vulnerable, are presented to underline the issue with the test. These are the Redmi Note 9 (device no. 7) and the Asus Zenfone 6 (device no. 8). For these 2 devices, the results stated that they were not installing the provided RSC, as shown in Figure 36 for the Redmi Note 9 device. Looking at the Wireshark capture of Test 7, Figure 37 shows that the Redmi Note 9 phone indeed reinstalls an all-zero RSC value, while it should take the counter value given by the AP.

```

[09:11:43] Renewed the group key
[09:11:44] 50:3d:c6:54:0e:15: Send group message 1 for the same GTK and max RSC
[09:11:44] 50:3d:c6:54:0e:15: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 2 ARPs this interval)
[09:11:45] 50:3d:c6:54:0e:15: received a new group message 2
[09:11:45] Renewed the group key
[09:11:46] 50:3d:c6:54:0e:15: Send group message 1 for the same GTK and max RSC
[09:11:46] 50:3d:c6:54:0e:15: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 3 ARPs this interval)
[09:11:46] 50:3d:c6:54:0e:15: received a new group message 2
[09:11:47] Renewed the group key
[09:11:49] 50:3d:c6:54:0e:15: Send group message 1 for the same GTK and max RSC
[09:11:49] 50:3d:c6:54:0e:15: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 4 ARPs this interval)
[09:11:49] 50:3d:c6:54:0e:15: received a new group message 2
[09:11:49] 50:3d:c6:54:0e:15: Client always installs the group key in the group key handshake with a zero replay counter (this is bad).
[09:11:49] Or client accepts replayed broadcast frames (see --replay-broadcast).
[09:11:50] Renewed the group key

```

Figure 36: Test-positive result for the Redmi Note 9 (Test 7)

| No. | Time         | Source            | Destination       | Protocol | Length | Replay Counter | WPA Key RSC      | CCMP Ext. Initialization Vector | Key Index | Info                       |
|-----|--------------|-------------------|-------------------|----------|--------|----------------|------------------|---------------------------------|-----------|----------------------------|
| 181 | 6.179151413  | Shenzhen_2f:f6:f0 | XiaomiCo_54:0e:15 | EAPOL    | 146    | 1              | 0000000000000000 |                                 |           | Key (Message 1 of 4)       |
| 182 | 6.181098677  | XiaomiCo_54:0e:15 | Shenzhen_2f:f6:f0 | EAPOL    | 195    | 1              | 0000000000000000 |                                 |           | Key (Message 2 of 4)       |
| 183 | 6.183985327  | Shenzhen_2f:f6:f0 | XiaomiCo_54:0e:15 | EAPOL    | 202    | 2              | 0000000000000000 |                                 |           | Key (Message 3 of 4)       |
| 185 | 6.187125389  | XiaomiCo_54:0e:15 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 2              | 0000000000000000 |                                 |           | Key (Message 4 of 4)       |
| 293 | 8.227557091  | Shenzhen_2f:f6:f0 | XiaomiCo_54:0e:15 | EAPOL    | 194    | 3              | ffff020000000000 | 0x000000000000                  | 0         | Key (Group Message 1 of 2) |
| 296 | 8.230767751  | XiaomiCo_54:0e:15 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 3              | 0000000000000000 | 0x000000000022                  | 0         | Key (Group Message 2 of 2) |
| 344 | 10.518659686 | Shenzhen_2f:f6:f0 | XiaomiCo_54:0e:15 | EAPOL    | 194    | 4              | ffff020000000000 | 0x000000000000                  | 0         | Key (Group Message 1 of 2) |
| 345 | 10.523833961 | XiaomiCo_54:0e:15 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 4              | 0000000000000000 | 0x000000000028                  | 0         | Key (Group Message 2 of 2) |
| 411 | 12.961412422 | Shenzhen_2f:f6:f0 | XiaomiCo_54:0e:15 | EAPOL    | 194    | 5              | ffff020000000000 | 0x000000000014                  | 0         | Key (Group Message 1 of 2) |
| 414 | 12.963958307 | XiaomiCo_54:0e:15 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 5              | 0000000000000000 | 0x00000000003A                  | 0         | Key (Group Message 2 of 2) |

Figure 37: Constant RSC reset to 0 by Redmi Note 9 during the group key handshake (Test 7)

The same was observed for device no. 8 (Figure 38), which would confirm the script’s output. Overall, it seems that the script output for Test 7 is less reliable given the various Wireshark captures taken. Nevertheless, the traffic captures seem to indicate that devices don’t install the provided RSC value in the group key handshake.

| No. | Time         | Source            | Destination       | Protocol | Length | Replay Counter | WPA Key RSC      | CCMP Ext. Initialization Vector | Key Index | Info                       |
|-----|--------------|-------------------|-------------------|----------|--------|----------------|------------------|---------------------------------|-----------|----------------------------|
| 305 | 11.777684153 | Shenzhen_2f:f6:f0 | 7e:b7:c3:2f:dc:05 | EAPOL    | 146    | 1              | 0000000000000000 |                                 |           | Key (Message 1 of 4)       |
| 308 | 11.812229563 | 7e:b7:c3:2f:dc:05 | Shenzhen_2f:f6:f0 | EAPOL    | 195    | 1              | 0000000000000000 |                                 |           | Key (Message 2 of 4)       |
| 310 | 11.818113556 | Shenzhen_2f:f6:f0 | 7e:b7:c3:2f:dc:05 | EAPOL    | 202    | 2              | 0000000000000000 |                                 |           | Key (Message 3 of 4)       |
| 312 | 11.822988449 | 7e:b7:c3:2f:dc:05 | Shenzhen_2f:f6:f0 | EAPOL    | 173    | 2              | 0000000000000000 |                                 |           | Key (Message 4 of 4)       |
| 500 | 14.435577415 | Shenzhen_2f:f6:f0 | 7e:b7:c3:2f:dc:05 | EAPOL    | 194    | 3              | ffff020000000000 | 0x000000000008                  | 0         | Key (Group Message 1 of 2) |
| 501 | 14.440847336 | Shenzhen_2f:f6:f0 | 7e:b7:c3:2f:dc:05 | EAPOL    | 189    | 3              | 0000000000000000 | 0x00000000002A                  | 0         | Key (Group Message 2 of 2) |
| 580 | 16.543574348 | 7e:b7:c3:2f:dc:05 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 4              | 0000000000000000 | 0x000000000034                  | 0         | Key (Group Message 2 of 2) |
| 591 | 16.558610586 | Shenzhen_2f:f6:f0 | 7e:b7:c3:2f:dc:05 | EAPOL    | 194    | 4              | ffff020000000000 | 0x000000000000                  | 0         | Key (Group Message 1 of 2) |
| 700 | 18.697649874 | Shenzhen_2f:f6:f0 | 7e:b7:c3:2f:dc:05 | EAPOL    | 194    | 5              | ffff020000000000 | 0x000000000015                  | 0         | Key (Group Message 1 of 2) |
| 701 | 18.703557896 | 7e:b7:c3:2f:dc:05 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 5              | 0000000000000000 | 0x000000000042                  | 0         | Key (Group Message 2 of 2) |
| 795 | 20.606475495 | Shenzhen_2f:f6:f0 | 7e:b7:c3:2f:dc:05 | EAPOL    | 194    | 6              | ffff020000000000 | 0x000000000017                  | 0         | Key (Group Message 1 of 2) |
| 799 | 20.639225118 | 7e:b7:c3:2f:dc:05 | Shenzhen_2f:f6:f0 | EAPOL    | 189    | 6              | 0000000000000000 | 0x000000000047                  | 0         | Key (Group Message 2 of 2) |

Figure 38: RSC is also not incremented in the group key handshake for Asus Zenfone 6 (Test 7)

### 8.2.3 KRACK against Samsung S5 (Android v6)

Another quite interesting case study is the Samsung S5 device (device no. 4). It was almost positive on all tests, which was less surprising than for other devices as its OS is Android v6.0.1. This OS was the one at the centre of Vanhoef's analyses, which he found to be vulnerable to KRACK [5, p. 5]. Starting with the general Test 1, PTK-IV reuse was detected, indicating PTK reinstallation (Figure 39). As opposed to the Blackberry device, the Samsung S5 was also reinstalling the GTK during Test 1 (Figure 40).

```
[09:00:48] Reset PN for GTK
[09:00:48] 60:af:6d:f9:69:c0: sending a new 4-way message 3 where the GTK has a zero RSC
[09:00:48] 60:af:6d:f9:69:c0: received a new message 4
[09:00:49] 60:af:6d:f9:69:c0: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 1 ARPs this interval)
[09:00:49] 60:af:6d:f9:69:c0: transmitted data using IV=79 (seq=5)
[09:00:49] 60:af:6d:f9:69:c0: no pairwise IV resets seem to have occurred for one interval
[09:00:49] 60:af:6d:f9:69:c0: transmitted data using IV=1 (seq=19)
[09:00:49] 60:af:6d:f9:69:c0: IV reuse detected (IV=1, seq=19). Client reinstalls the pairwise key in the 4-way handshake (this is bad)
[09:00:49] 60:af:6d:f9:69:c0: transmitted data using IV=2 (seq=20)
[09:00:49] 60:af:6d:f9:69:c0: transmitted data using IV=3 (seq=21)
[09:00:49] 60:af:6d:f9:69:c0: transmitted data using IV=4 (seq=22)
```

Figure 39: Samsung S5 being vulnerable to KRACK and performing PTK reinstallation (Test 1)

```
[09:01:08] Reset PN for GTK
[09:01:08] 60:af:6d:f9:69:c0: sending a new 4-way message 3 where the GTK has a zero RSC
[09:01:09] 60:af:6d:f9:69:c0: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 3 ARPs this interval)
[09:01:09] 60:af:6d:f9:69:c0: transmitted data using IV=42 (seq=29)
[09:01:09] 60:af:6d:f9:69:c0: transmitted data using IV=50 (seq=31)
[09:01:09] 60:af:6d:f9:69:c0: received 5 replies to the replayed broadcast ARP requests
[09:01:09] 60:af:6d:f9:69:c0: Client reinstalls the group key in the 4-way handshake (this is bad).
[09:01:09] Or client accepts replayed broadcast frames (see --replay-broadcast).
[09:01:10] 60:af:6d:f9:69:c0: transmitted data using IV=67 (seq=32)
[09:01:10] Reset PN for GTK
```

Figure 40: GTK reinstallation in the 4-way handshake for device no. 4 (Test 1)

The Wireshark capture confirms that a PTK reinstallation occurred in frame no. 7452, after the second retransmission of EAPOL Message 3 (Figure 41). Note that for this device, the group key was also reinstalled, as the Samsung S5 device replied to replayed ARP requests multiple times (see Figure 42 on p. 67). The *tptk* variant of KRACK (Test 3) also produced the same outcomes for device no. 4. Still, although Vanhoef found that Android v6.0.1 devices were reinstalling an all-zero TK, the captured frames showed that device no. 4 wasn't.

| No.  | Time         | Source                    | Destination       | Protocol | Length | Replay | WPA Key RSC      | CCMP Ext. Initialization Vector | Key Index | Info                                                 |
|------|--------------|---------------------------|-------------------|----------|--------|--------|------------------|---------------------------------|-----------|------------------------------------------------------|
| 6686 | 37.388476182 | Shenzhen 2f:f6:f0         | SamsungE_f9:69:c0 | EAPOL    | 146    | 1      | 0000000000000000 | 0x0000000000000000              | 0         | Key (Message 1 of 4)                                 |
| 6688 | 37.389436883 | SamsungE_f9:69:c0         | Shenzhen 2f:f6:f0 | EAPOL    | 195    | 1      | 0000000000000000 | 0x0000000000000000              | 0         | Key (Message 2 of 4)                                 |
| 6691 | 37.392782391 | Shenzhen 2f:f6:f0         | SamsungE_f9:69:c0 | EAPOL    | 292    | 2      | 0000000000000000 | 0x0000000000000000              | 0         | Key (Message 3 of 4)                                 |
| 6693 | 37.405317389 | SamsungE_f9:69:c0         | Shenzhen 2f:f6:f0 | EAPOL    | 173    | 2      | 0000000000000000 | 0x0000000000000000              | 0         | Key (Message 4 of 4)                                 |
| 6798 | 37.633443311 | ::                        | ff02::1:ff9:69:c0 | ICMPv6   | 154    |        |                  | 0x0000000000000001              | 0         | Neighbor Solicitation for fe80::62af:6dff:fe99:69c0  |
| 6799 | 37.633936275 | ::                        | ff02::1:ff9:69:c0 | ICMPv6   | 168    |        |                  | 0x0000000000000002              | 0         | Multicast Listener Report Message V2                 |
| 6845 | 38.117189104 | fe80::62af:6dff:fe99:69c0 | ff02::1:ff9:69:c0 | ICMPv6   | 146    |        |                  | 0x0000000000000004              | 0         | Router Solicitation from 60:af:6d:f9:69:c0           |
| 6851 | 38.157687837 | 0.0.0.0                   | 255.255.255.255   | DHCP     | 432    |        |                  | 0x0000000000000005              | 0         | DHCP Request - Transaction ID 0xc1dd2d77             |
| 6902 | 38.616948325 | fe80::62af:6dff:fe99:69c0 | ff02::1:ff9:69:c0 | ICMPv6   | 146    |        |                  | 0x0000000000000006              | 0         | Router Solicitation from 60:af:6d:f9:69:c0           |
| 6926 | 38.809991821 | SamsungE_f9:69:c0         | Broadcast         | ARP      | 118    |        |                  | 0x0000000000000007              | 0         | Who has 192.168.100.254? Tell 192.168.100.2          |
| 6930 | 38.814181576 | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |        |                  | 0x0000000000000008              | 0         | Standard query 0x17b1 A connectivitycheck.android.c  |
| 6931 | 38.814244610 | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |        |                  | 0x0000000000000009              | 0         | Standard query 0x7239 A connectivitycheck.android.c  |
| 7012 | 39.109265013 | 192.168.100.2             | 192.168.100.254   | DNS      | 162    |        |                  | 0x000000000000000e              | 0         | Standard query 0xc8f A north-america.pool.ntp.org    |
| 7019 | 39.219850327 | 192.168.100.2             | 192.168.100.254   | DNS      | 162    |        |                  | 0x0000000000000010              | 0         | Standard query 0xc8f A north-america.pool.ntp.org    |
| 7022 | 39.223964395 | 192.168.100.2             | 192.168.100.254   | DNS      | 178    |        |                  | 0x0000000000000011              | 0         | Standard query 0xdf49 A north-america.pool.ntp.org   |
| 7023 | 39.224016051 | 192.168.100.2             | 192.168.100.254   | DNS      | 162    |        |                  | 0x0000000000000012              | 0         | Standard query 0xe359 A north-america.pool.ntp.org   |
| 7028 | 39.231433165 | 192.168.100.2             | 192.168.100.254   | DNS      | 178    |        |                  | 0x0000000000000013              | 0         | Standard query 0xdf49 A north-america.pool.ntp.org   |
| 7031 | 39.233795957 | 192.168.100.2             | 192.168.100.254   | DNS      | 162    |        |                  | 0x0000000000000014              | 0         | Standard query 0xe359 A north-america.pool.ntp.org   |
| 7077 | 39.611993430 | fe80::62af:6dff:fe99:69c0 | ff02::1:ff9:69:c0 | ICMPv6   | 146    |        |                  | 0x000000000000001d              | 0         | Router Solicitation from 60:af:6d:f9:69:c0           |
| 7083 | 39.670917432 | Shenzhen 2f:f6:f0         | SamsungE_f9:69:c0 | EAPOL    | 210    | 3      | 0000000000000000 | 0x0000000000000020              | 0         | Key (Message 3 of 4)                                 |
| 7092 | 39.714887739 | SamsungE_f9:69:c0         | Shenzhen 2f:f6:f0 | EAPOL    | 189    | 3      | 0000000000000000 | 0x0000000000000028              | 0         | Key (Message 4 of 4)                                 |
| 7208 | 40.326764647 | 192.168.100.2             | 192.168.100.254   | DNS      | 158    |        |                  | 0x000000000000002d              | 0         | Standard query 0x898a A mqtt-mini.facebook.com       |
| 7213 | 40.330655134 | 192.168.100.2             | 192.168.100.254   | DNS      | 158    |        |                  | 0x000000000000002e              | 0         | Standard query 0x898a A mqtt-mini.facebook.com       |
| 7237 | 40.520688416 | 192.168.100.2             | 192.168.100.254   | DNS      | 172    |        |                  | 0x0000000000000033              | 0         | Standard query 0xdf4b A firebaseinstallations.google |
| 7253 | 40.619271490 | fe80::62af:6dff:fe99:69c0 | ff02::1:ff9:69:c0 | ICMPv6   | 146    |        |                  | 0x0000000000000036              | 0         | Router Solicitation from 60:af:6d:f9:69:c0           |
| 7287 | 40.833913294 | SamsungE_f9:69:c0         | Shenzhen 2f:f6:f0 | ARP      | 118    |        |                  | 0x0000000000000038              | 0         | 192.168.100.2 is at 60:af:6d:f9:69:c0                |
| 7304 | 41.640686990 | fe80::62af:6dff:fe99:69c0 | ff02::1:ff9:69:c0 | ICMPv6   | 146    |        |                  | 0x000000000000004c              | 0         | Router Solicitation from 60:af:6d:f9:69:c0           |
| 7481 | 41.743184139 | Shenzhen 2f:f6:f0         | SamsungE_f9:69:c0 | EAPOL    | 218    | 4      | 0000000000000000 | 0x000000000000005b              | 0         | Key (Message 3 of 4)                                 |
| 7484 | 41.751801115 | SamsungE_f9:69:c0         | Shenzhen 2f:f6:f0 | EAPOL    | 189    | 4      | 0000000000000000 | 0x000000000000006c              | 0         | Key (Message 4 of 4)                                 |
| 7452 | 42.151998844 | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |        |                  | 0x0000000000000001              | 0         | Standard query 0x469d A connectivitycheck.android.c  |
| 7453 | 42.152093246 | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |        |                  | 0x0000000000000002              | 0         | Standard query 0xadad A connectivitycheck.android.c  |
| 7484 | 42.354598350 | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |        |                  | 0x0000000000000003              | 0         | Standard query 0x4855 A connectivitycheck.android.c  |
| 7485 | 42.354575842 | 192.168.100.2             | 192.168.100.254   | DNS      | 165    |        |                  | 0x0000000000000004              | 0         | Standard query 0x5173 A connectivitycheck.android.c  |
| 7497 | 42.487335562 | fe80::62af:6dff:fe99:69c0 | ff02::1:ff9:69:c0 | ICMPv6   | 146    |        |                  | 0x0000000000000005              | 0         | Router Solicitation from 60:af:6d:f9:69:c0           |

Figure 41: IV reuse highlighted due to PTK reinstallation on Samsung S5 (Test 1)



| No.   | Time         | Source             | Destination        | Protocol | Length | Replay | WPA Key RSC | CCMP Ext. Initialization Vector | Key Index | Info                                          |
|-------|--------------|--------------------|--------------------|----------|--------|--------|-------------|---------------------------------|-----------|-----------------------------------------------|
| 6927  | 38.811986310 | Samsung_E_f9:69:c0 | Broadcast          | ARP      | 89     |        |             | 0x000000000004                  |           | 1 Who has 192.168.100.254? Tell 192.168.100.2 |
| 6929  | 38.812642250 | Shenzhen_2f:f6:f0  | Samsung_E_f9:69:c0 | ARP      | 91     |        |             | 0x000000000002                  |           | 0 192.168.100.254 is at 68:af:6d:f9:69:c0     |
| 7286  | 40.832935515 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000002                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 7287  | 40.833913294 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x000000000038                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0       |
| 7533  | 42.752047916 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 7534  | 42.753166920 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x000000000007                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0       |
| 7791  | 44.929531286 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 8088  | 46.980457995 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 8017  | 46.991266123 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x00000000001F                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0       |
| 8337  | 49.827708117 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 8338  | 49.828809780 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x000000000029                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0       |
| 8612  | 51.078022123 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 8613  | 51.079086705 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x000000000038                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0       |
| 8831  | 53.063908942 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 8834  | 53.06515532  | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x000000000049                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0       |
| 9597  | 55.078380837 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000002                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 9598  | 55.080950342 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x00000000007C                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0       |
| 9837  | 56.878319682 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x000000000007                  |           | 0 Who has 192.168.100.254? Tell 192.168.100.2 |
| 9895  | 57.223912140 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 9960  | 57.878078139 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x000000000011                  |           | 0 Who has 192.168.100.254? Tell 192.168.100.2 |
| 10086 | 58.878247666 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x00000000001C                  |           | 0 Who has 192.168.100.254? Tell 192.168.100.2 |
| 10119 | 59.160906083 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x00000000001F                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0       |
| 10120 | 59.161476622 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 10224 | 60.087602952 | Samsung_E_f9:69:c0 | Broadcast          | ARP      | 118    |        |             | 0x000000000022                  |           | 0 Who has 192.168.100.254? Tell 192.168.100.2 |
| 10362 | 61.084316443 | Samsung_E_f9:69:c0 | Broadcast          | ARP      | 118    |        |             | 0x000000000025                  |           | 0 Who has 192.168.100.254? Tell 192.168.100.2 |
| 10378 | 61.198291786 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 10379 | 61.200154306 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x000000000026                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0       |
| 10550 | 62.084182956 | Samsung_E_f9:69:c0 | Broadcast          | ARP      | 118    |        |             | 0x000000000028                  |           | 0 Who has 192.168.100.254? Tell 192.168.100.2 |
| 10715 | 63.167559375 | Samsung_E_f9:69:c0 | Broadcast          | ARP      | 118    |        |             | 0x00000000002D                  |           | 0 Who has 192.168.100.254? Tell 192.168.100.2 |
| 10742 | 63.369643776 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |        |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1   |
| 10743 | 63.370554334 | Samsung_E_f9:69:c0 | Shenzhen_2f:f6:f0  | ARP      | 118    |        |             | 0x000000000037                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0       |

Figure 42: Device no. 4 repeatedly replies to ARP requests, indicating GTK reinstallation (Test 1)

Responses to replayed ARP requests were also detected during the group key handshake (Figure 43) and proven during traffic capture (Figure 44). Note that in Figure 44, all ARP frames or frames which were encrypted by the AP (i.e., “Key Index” column shows “1”) are filtered for. Even if the client’s IP address is sometimes shown, it is the AP who retransmitted the client’s broadcast frame. If just looking at ARP frames isn’t enough to identify GTK reinstallation, this display can help follow the GTK-IVs more precisely and show that high-lighted IVs were already used previously by the AP.

```
[09:09:51] Renewed the group key
[09:09:52] 60:af:6d:f9:69:c0: Send group message 1 for the same GTK and max RSC
[09:09:52] 60:af:6d:f9:69:c0: sending broadcast ARP to 192.168.100.2 from 192.168.100.1 (sent 2 ARPs this interval)
[09:09:52] 60:af:6d:f9:69:c0: transmitted data using IV=173 (seq=86)
[09:09:52] 60:af:6d:f9:69:c0: received a new group message 2
[09:09:52] 60:af:6d:f9:69:c0: transmitted data using IV=182 (seq=87)
[09:09:52] 60:af:6d:f9:69:c0: transmitted data using IV=183 (seq=88)
[09:09:52] 60:af:6d:f9:69:c0: transmitted data using IV=184 (seq=89)
[09:09:52] 60:af:6d:f9:69:c0: transmitted data using IV=185 (seq=90)
[09:09:52] 60:af:6d:f9:69:c0: transmitted data using IV=190 (seq=16)
[09:09:52] 60:af:6d:f9:69:c0: transmitted data using IV=190 (seq=91)
[09:09:52] 60:af:6d:f9:69:c0: transmitted data using IV=191 (seq=92)
[09:09:52] 60:af:6d:f9:69:c0: received 5 replies to the replayed broadcast ARP requests
[09:09:52] 60:af:6d:f9:69:c0: Client always installs the group key in the group key handshake with a zero replay counter (this is bad)
[09:09:52] Or client accepts replayed broadcast frames (see --replay-broadcast).
```

Figure 43: Samsung S5 reinstalls the GTK during the group key handshake (Test 5)

| No.  | Time         | Source                    | Destination        | Protocol | Length | Replay Counter | WPA Key RSC | CCMP Ext. Initialization Vector | Key Index | Info                                            |
|------|--------------|---------------------------|--------------------|----------|--------|----------------|-------------|---------------------------------|-----------|-------------------------------------------------|
| 1542 | 8.801941212  | 192.168.100.2             | 224.0.0.22         | ICMPv3   | 161    |                |             | 0x000000000002                  |           | 1 Membership Report / Join group 224.0.0.251 fo |
| 1612 | 8.556707545  | 192.168.100.2             | 224.0.0.251        | NDNS     | 336    |                |             | 0x000000000003                  |           | 1 Standard query response 0x0000 PTR, cache fl  |
| 1613 | 8.558927702  | fe80::62af:6dff:fe9:69c0  | ff02::fb           | NDNS     | 356    |                |             | 0x000000000004                  |           | 1 Standard query response 0x0000 PTR, cache fl  |
| 1664 | 9.016903559  | Shenzhen_2f:f6:f0         | Broadcast          | ARP      | 89     |                |             | 0x000000000005                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1     |
| 1877 | 10.550440893 | 192.168.100.2             | 224.0.0.251        | NDNS     | 336    |                |             | 0x000000000001                  |           | 1 Standard query response 0x0000 PTR, cache fl  |
| 1879 | 10.564081542 | fe80::62af:6dff:fe9:69c0  | ff02::fb           | NDNS     | 356    |                |             | 0x000000000002                  |           | 1 Standard query response 0x0000 PTR, cache fl  |
| 1912 | 10.847701493 | Samsung_E_f9:69:c0        | Shenzhen_2f:f6:f0  | ARP      | 118    |                |             | 0x00000000004E                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0         |
| 1913 | 10.847729767 | Shenzhen_2f:f6:f0         | Broadcast          | ARP      | 89     |                |             | 0x000000000003                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1     |
| 2021 | 11.545361182 | fe80::62af:6dff:fe9:69c0  | ff02::12           | ICMPv6   | 117    |                |             | 0x000000000004                  |           | 1 Router Solicitation from 68:af:6d:f9:69:c0    |
| 2115 | 12.297801141 | fe80::c1c1:38ff:fe2f:f6f0 | ff02::12           | ICMPv6   | 117    |                |             | 0x000000000001                  |           | 1 Router Solicitation from c1c1:38ff:fe2f:f6f0  |
| 2359 | 12.904872502 | Shenzhen_2f:f6:f0         | Broadcast          | ARP      | 89     |                |             | 0x000000000002                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1     |
| 2360 | 12.905734719 | Samsung_E_f9:69:c0        | Shenzhen_2f:f6:f0  | ARP      | 118    |                |             | 0x000000000066                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0         |
| 2756 | 14.566673675 | 192.168.100.2             | 224.0.0.251        | NDNS     | 336    |                |             | 0x000000000001                  |           | 1 Standard query response 0x0000 PTR, cache fl  |
| 2757 | 14.569612938 | fe80::62af:6dff:fe9:69c0  | ff02::fb           | NDNS     | 356    |                |             | 0x000000000002                  |           | 1 Standard query response 0x0000 PTR, cache fl  |
| 2838 | 15.163575599 | Shenzhen_2f:f6:f0         | Broadcast          | ARP      | 89     |                |             | 0x000000000003                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1     |
| 2839 | 15.164455828 | Samsung_E_f9:69:c0        | Shenzhen_2f:f6:f0  | ARP      | 118    |                |             | 0x000000000005                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0         |
| 2842 | 15.172837505 | fe80::62af:6dff:fe9:69c0  | ff02::16           | ICMPv6   | 137    |                |             | 0x000000000004                  |           | 1 Multicast Listener Report Message v2          |
| 2890 | 15.559335287 | fe80::62af:6dff:fe9:69c0  | ff02::12           | ICMPv6   | 117    |                |             | 0x000000000005                  |           | 1 Router Solicitation from 68:af:6d:f9:69:c0    |
| 3077 | 17.013117151 | Shenzhen_2f:f6:f0         | Broadcast          | ARP      | 89     |                |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1     |
| 3078 | 17.016319127 | Samsung_E_f9:69:c0        | Shenzhen_2f:f6:f0  | ARP      | 118    |                |             | 0x000000000006                  |           | 0 192.168.100.2 is at 68:af:6d:f9:69:c0         |
| 3427 | 19.261244572 | Shenzhen_2f:f6:f0         | Broadcast          | ARP      | 89     |                |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1     |
| 3699 | 21.311821575 | Shenzhen_2f:f6:f0         | Broadcast          | ARP      | 89     |                |             | 0x000000000001                  |           | 1 Who has 192.168.100.2? Tell 192.168.100.1     |
| 3810 | 22.125642175 | 192.168.100.2             | 224.0.0.22         | ICMPv3   | 161    |                |             | 0x000000000002                  |           | 1 Membership Report / Join group 224.0.0.251 fo |
| 3812 | 22.128476053 | fe80::62af:6dff:fe9:69c0  | ff02::16           | ICMPv6   | 137    |                |             | 0x000000000003                  |           | 1 Multicast Listener Report Message v2          |
| 3898 | 22.610383390 | 192.168.100.2             | 224.0.0.251        | NDNS     | 170    |                |             | 0x000000000001                  |           | 1 Standard query 0x0000 ANY Android.local, "Q"  |
| 3900 | 22.612494810 | fe80::62af:6dff:fe9:69c0  | ff02::fb           | NDNS     | 190    |                |             | 0x000000000002                  |           | 1 Standard query 0x0000 ANY Android.local, "Q"  |
| 3929 | 22.846390128 | Samsung_E_f9:69:c0        | Shenzhen_2f:f6:f0  | ARP      | 118    |                |             | 0x000000000013                  |           | 0 Who has 192.168.100.254? Tell 192.168.100.2   |
| 3933 | 22.849652201 | Shenzhen_2f:f6:f0         | Samsung_E_f9:69:c0 | ARP      | 91     |                |             | 0x000000000004                  |           | 0 192.168.100.254 is at c1c1:38ff:fe2f:f6f0     |

Figure 44: Multiple ARP responses to replayed ARP requests by device no. 4 (Test 5)

Out of all 29 devices, the Samsung S5 seems therefore to be the most vulnerable to KRACK attacks.

### 8.2.3 Replaying Broadcast Frames on Surface 3

The last example presented for this subchapter is the Surface 3 tablet (device no. 1), the only device where Test 2 came out positive. It should be remembered that the test looks for replies to replayed ARP messages based on GTK-IVs. For this test, no KRACK attempts are performed (meaning that no GTK is reinstalled, unlike for the other options). The Surface 3 tablet replied to a replayed broadcasted ARP request, as can be seen in [Figure 45](#). The replayed frames are highlighted in red, while the response is marked in green.

| No.  | Time         | Source             | Destination        | Protocol | Length | Replay Counter | CCMP Est. Initialization Vector | Key Index | Info                                            |
|------|--------------|--------------------|--------------------|----------|--------|----------------|---------------------------------|-----------|-------------------------------------------------|
| 839  | 10.552423593 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 89     |                | 0x000000000004                  |           | 1 who has 192.168.100.254? Tell 169.254.235.118 |
| 840  | 10.554331802 | Shenzhen_2f:f6:f0  | Microsoft_2c:b7:e9 | ARP      | 91     |                | 0x000000000007                  |           | 0 192.168.100.254 is at c8:1c:30:2f:f6:f0       |
| 927  | 12.368654911 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |                | 0x000000000001                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1     |
| 1056 | 14.111997284 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |                | 0x000000000002                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1     |
| 1101 | 14.306064142 | Microsoft_2c:b7:e9 | Shenzhen_2f:f6:f0  | ARP      | 118    |                | 0x00000000007f                  |           | 0 192.168.100.2 is at b4:ae:2b:2c:b7:e9         |
| 1102 | 14.306106055 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |                | 0x00000000000f                  |           | 1 who has 192.168.100.2? Tell 192.168.100.254   |
| 1163 | 14.460802892 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 118    |                | 0x000000000094                  |           | 0 who has 192.168.100.2? (ARP Probe)            |
| 1168 | 14.470770279 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 89     |                | 0x000000000027                  |           | 1 who has 192.168.100.2? (ARP Probe)            |
| 1205 | 14.918521339 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 118    |                | 0x00000000009d                  |           | 0 who has 192.168.100.254? Tell 192.168.100.2   |
| 1216 | 14.923426519 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 89     |                | 0x00000000002a                  |           | 1 who has 192.168.100.254? Tell 192.168.100.2   |
| 1217 | 14.924599970 | Shenzhen_2f:f6:f0  | Microsoft_2c:b7:e9 | ARP      | 91     |                | 0x000000000011                  |           | 0 192.168.100.254 is at c8:1c:30:2f:f6:f0       |
| 1270 | 15.469125609 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 118    |                | 0x0000000000a8                  |           | 0 who has 192.168.100.2? (ARP Probe)            |
| 1279 | 15.473703926 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 89     |                | 0x000000000001                  |           | 1 who has 192.168.100.2? (ARP Probe)            |
| 1341 | 16.149092053 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |                | 0x000000000005                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1     |
| 1342 | 16.151018408 | Microsoft_2c:b7:e9 | Shenzhen_2f:f6:f0  | ARP      | 118    |                | 0x0000000000b4                  |           | 0 192.168.100.2 is at b4:ae:2b:2c:b7:e9         |
| 1385 | 16.479253386 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 118    |                | 0x0000000000b8                  |           | 0 who has 192.168.100.2? (ARP Probe)            |
| 1386 | 16.483186656 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 89     |                | 0x000000000006                  |           | 1 who has 192.168.100.2? (ARP Probe)            |
| 1467 | 17.471565541 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 118    |                | 0x0000000000c3                  |           | 0 ARP Announcement for 192.168.100.2            |
| 1469 | 17.483980491 | Microsoft_2c:b7:e9 | Broadcast          | ARP      | 89     |                | 0x000000000001                  |           | 1 ARP Announcement for 192.168.100.2            |
| 1515 | 18.208119484 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |                | 0x000000000003                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1     |
| 1516 | 18.209020339 | Microsoft_2c:b7:e9 | Shenzhen_2f:f6:f0  | ARP      | 118    |                | 0x0000000000cf                  |           | 0 192.168.100.2 is at b4:ae:2b:2c:b7:e9         |
| 1883 | 19.979026775 | Microsoft_2c:b7:e9 | Shenzhen_2f:f6:f0  | ARP      | 118    |                | 0x000000000150                  |           | 0 who has 192.168.100.254? Tell 192.168.100.2   |
| 1884 | 19.981389138 | Shenzhen_2f:f6:f0  | Microsoft_2c:b7:e9 | ARP      | 91     |                | 0x00000000002e                  |           | 0 192.168.100.254 is at c8:1c:30:2f:f6:f0       |
| 1929 | 20.559630712 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |                | 0x000000000011                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1     |
| 1980 | 21.697184595 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |                | 0x000000000002                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1     |
| 2211 | 24.649911147 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |                | 0x000000000001                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1     |
| 2215 | 24.663715202 | Microsoft_2c:b7:e9 | Shenzhen_2f:f6:f0  | ARP      | 118    |                | 0x000000000189                  |           | 0 192.168.100.2 is at b4:ae:2b:2c:b7:e9         |
| 2386 | 26.396368193 | Microsoft_2c:b7:e9 | Shenzhen_2f:f6:f0  | ARP      | 118    |                | 0x0000000001ac                  |           | 0 192.168.100.2 is at b4:ae:2b:2c:b7:e9         |
| 2387 | 26.396381625 | Broadcast          | Broadcast          | ARP      | 89     |                | 0x000000000008                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1     |
| 2593 | 28.437859595 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |                | 0x000000000000                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1     |
| 2596 | 28.443379209 | Microsoft_2c:b7:e9 | Shenzhen_2f:f6:f0  | ARP      | 118    |                | 0x0000000001da                  |           | 0 192.168.100.2 is at b4:ae:2b:2c:b7:e9         |
| 2731 | 30.498111313 | Shenzhen_2f:f6:f0  | Broadcast          | ARP      | 89     |                | 0x000000000007                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1     |
| 2732 | 30.498617428 | Microsoft_2c:b7:e9 | Shenzhen_2f:f6:f0  | ARP      | 118    |                | 0x0000000001ee                  |           | 0 192.168.100.2 is at b4:ae:2b:2c:b7:e9         |

Figure 45: A replayed ARP request accepted by device no. 1 (Test 2)

Since the Surface 3 device has seen (i.e., responded to) other ARP frames with higher GTK-IVs, it should have dropped frame no. 2211. Multiple examples can be seen in the capture. Interestingly, the device answered to other frames where the GTK-IV was lower than that of previously accepted ones. For example, frame no. 1341 used a GTK-IV of “5”, but it was a GTK-IV of “3” for frame no. 1515. Therefore, frame no. 1515 should have been dropped. Instead, it replied to it. [Figure 46](#) shows the GTK-IV incrementation of the AP, and highlights frames encrypted using GTK-IVs of “1” by the AP.

| No.  | Time         | Source                    | Destination        | Protocol | Length | Replay Counter | WPA Key RSC | CCMP Est. Initialization Vector | Key Index | Info                                                |
|------|--------------|---------------------------|--------------------|----------|--------|----------------|-------------|---------------------------------|-----------|-----------------------------------------------------|
| 1754 | 19.091491703 | 192.168.100.2             | 224.0.0.252        | LLMNR    | 111    |                |             | 0x00000000000b                  |           | 1 Standard query 0x460f A upad                      |
| 1755 | 19.092385851 | fe80::861b:c102:c22e:4430 | ff02::1:1:3        | LLMNR    | 131    |                |             | 0x00000000001c                  |           | 1 Standard query 0xc489 A upad                      |
| 1756 | 19.093241096 | 192.168.100.2             | 224.0.0.252        | LLMNR    | 111    |                |             | 0x00000000001d                  |           | 1 Standard query 0xc489 A upad                      |
| 1759 | 19.090824783 | fe80::861b:c102:c22e:4430 | ff02::1:1:3        | LLMNR    | 131    |                |             | 0x00000000001e                  |           | 1 Standard query 0x45ff A upad                      |
| 1760 | 19.101063701 | 192.168.100.2             | 224.0.0.252        | LLMNR    | 111    |                |             | 0x00000000001f                  |           | 1 Standard query 0x45ff A upad                      |
| 1820 | 19.420500300 | 192.168.100.2             | 192.168.100.255    | NBNS     | 139    |                |             | 0x000000000001                  |           | 1 Name query NB UPAD<0>                             |
| 1820 | 19.424517892 | 192.168.100.2             | 192.168.100.255    | NBNS     | 139    |                |             | 0x000000000002                  |           | 1 Name query NB UPAD<0>                             |
| 1829 | 19.435887882 | 192.168.100.2             | 192.168.100.255    | NBNS     | 139    |                |             | 0x000000000003                  |           | 1 Name query NB UPAD<0>                             |
| 1830 | 19.436791036 | 192.168.100.2             | 192.168.100.255    | NBNS     | 139    |                |             | 0x000000000004                  |           | 1 Name query NB UPAD<0>                             |
| 1849 | 19.605551485 | 192.168.100.2             | 224.0.0.251        | PDNS     | 117    |                |             | 0x000000000005                  |           | 1 Standard query 0x0000 A upad.local, "QM" question |
| 1850 | 19.607116435 | fe80::861b:c102:c22e:4430 | ff02::fb           | PDNS     | 137    |                |             | 0x000000000006                  |           | 1 Standard query 0x0000 A upad.local, "QM" question |
| 1851 | 19.669285020 | 192.168.100.2             | 224.0.0.251        | PDNS     | 117    |                |             | 0x000000000007                  |           | 1 Standard query 0x0000 A upad.local, "QM" question |
| 1852 | 19.672131612 | fe80::861b:c102:c22e:4430 | ff02::fb           | PDNS     | 137    |                |             | 0x000000000008                  |           | 1 Standard query 0x0000 A upad.local, "QM" question |
| 1857 | 19.682070322 | 192.168.100.2             | 224.0.0.251        | PDNS     | 117    |                |             | 0x000000000009                  |           | 1 Standard query 0x0000 A upad.local, "QM" question |
| 1858 | 19.684328310 | fe80::861b:c102:c22e:4430 | ff02::fb           | PDNS     | 137    |                |             | 0x00000000000a                  |           | 1 Standard query 0x0000 A upad.local, "QM" question |
| 1859 | 19.686442157 | 192.168.100.2             | 224.0.0.251        | PDNS     | 117    |                |             | 0x00000000000b                  |           | 1 Standard query 0x0000 A upad.local, "QM" question |
| 1860 | 19.688404940 | fe80::861b:c102:c22e:4430 | ff02::fb           | PDNS     | 137    |                |             | 0x00000000000c                  |           | 1 Standard query 0x0000 A upad.local, "QM" question |
| 1863 | 19.979026775 | Microsoft_2c:b7:e9        | Shenzhen_2f:f6:f0  | ARP      | 118    |                |             | 0x000000000150                  |           | 0 who has 192.168.100.254? Tell 192.168.100.2       |
| 1884 | 19.981389138 | Shenzhen_2f:f6:f0         | Microsoft_2c:b7:e9 | ARP      | 91     |                |             | 0x00000000002e                  |           | 0 192.168.100.254 is at c8:1c:30:2f:f6:f0           |
| 1896 | 20.172956727 | 192.168.100.2             | 192.168.100.255    | NBNS     | 139    |                |             | 0x000000000000                  |           | 1 Name query NB UPAD<0>                             |
| 1900 | 20.185200286 | 192.168.100.2             | 192.168.100.255    | NBNS     | 139    |                |             | 0x000000000000                  |           | 1 Name query NB UPAD<0>                             |
| 1901 | 20.186296752 | 192.168.100.2             | 192.168.100.255    | NBNS     | 139    |                |             | 0x000000000000                  |           | 1 Name query NB UPAD<0>                             |
| 1902 | 20.188234282 | 192.168.100.2             | 192.168.100.255    | NBNS     | 139    |                |             | 0x000000000010                  |           | 1 Name query NB UPAD<0>                             |
| 1929 | 20.559630712 | Shenzhen_2f:f6:f0         | Broadcast          | ARP      | 89     |                |             | 0x000000000011                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1         |
| 1996 | 21.729015012 | fe80::c21c:30ff:fe2f:f6f0 | ff02::c            | ICMPv6   | 117    |                |             | 0x000000000001                  |           | 1 Router Solicitation from c8:1c:30:2f:f6:f0        |
| 2000 | 22.087344595 | Shenzhen_2f:f6:f0         | Broadcast          | ARP      | 89     |                |             | 0x000000000002                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1         |
| 2211 | 24.649911147 | Shenzhen_2f:f6:f0         | Broadcast          | ARP      | 89     |                |             | 0x000000000001                  |           | 1 who has 192.168.100.2? Tell 192.168.100.1         |
| 2214 | 24.661764952 | fe80::861b:c102:c22e:4430 | ff02::c            | UDP      | 765    |                |             | 0x000000000002                  |           | 1 56538 + 3702 Len=656                              |
| 2215 | 24.663715202 | Microsoft_2c:b7:e9        | Shenzhen_2f:f6:f0  | ARP      | 118    |                |             | 0x000000000189                  |           | 0 192.168.100.2 is at b4:ae:2b:2c:b7:e9             |
| 2218 | 24.668675635 | 192.168.100.2             | 239.255.255.250    | UDP      | 745    |                |             | 0x000000000003                  |           | 1 56537 + 3702 Len=656                              |

Figure 46: Expanded view of traffic with the Surface 3 device to follow GTK-IVs (Test 2)

### 8.3 Impact and Limits of Results

Having reviewed the relevant test cases, the significance of obtained results is discussed, considering the research questions. As a reminder, they are listed below:

- Q1. Given the identified available GitHub repositories, is it possible to successfully scan Wi-Fi-enabled devices for the KRACK vulnerability?
- Q2. If that is the case, what are the struggles to perform the vulnerability test?
- Q3. What's the current relevance of KRACK given obtained results with this tool?

Answering the first research question, it appears that a KRACK-scanning tool indeed allows for successful detection of the KRACK vulnerability on devices in most cases. Nevertheless, there are multiple reasons why it is imperfect. Note that the specific tests cited in the [Results](#) chapter are targeted here. To begin with, the script understandably can't handle all types of device behaviours: some devices break out of the connection, and it was shown that it is sensitive to background noise. Also, some obtained results were hard to verify, such as the tests resetting the RSC counter for the GTK (in the group key and 4-way handshakes). For some test, it would be correct while for others, the command line output would contradict the Wireshark capture. Although Vanhoef warns about their reliability issue on its GitHub repository [27], the exact reason behind them remains unclear. Undoubtedly, there was also a logical bug in Vanhoef's script which made the "gtkinit" option (Test 5) not set the RSC value properly in EAPOL Message 3. The impact of this, although being corrected afterwards, was unknown as it appeared that some devices were still stated as responding correctly while they weren't. A similar issue occurred for the "group-gtkinit" option (Test 7), while this one didn't suffer such a bug. For the main tests which target the most important vulnerabilities (Tests 1-3-4 and Test 6), the script was successfully identifying vulnerable devices. Overall, the script seemed efficient for marking devices which would be vulnerable to CVE-2017-13077, CVE-2017-13078 and CVE-2017-13080.

To answer the second research question, the following struggles for the testing process were therefore identified:

1. Finding a suitable WNIC for setting up the testing AP can be challenging.
2. The tests are sensitive to background noise.
3. Some device behaviour couldn't be handled by the script.
4. Some tests aren't performing correctly (Test 5 and Test 7).

Finally, the assessment of the KRACK vulnerabilities' current relevance, in light of the tests performed with the tool, answers the third and last research question. Given the comprehensive analyses presented in this thesis, it would seem that KRACK's main vulnerabilities were patched in modern OSs. Performing a KRACK attack against a victim to read its communications with an AP seems hard enough to deter attackers from choosing this attack vector though. First, an attacker would probably have to gain in-depth knowledge of how WPA/WPA2 works to fully recreate a tool able to exploit KRACK. Although the MC-MITM tool [47] and the all-zero key reinstallation Proof-of-Concept [48] written by Vanhoef are available, no tool or script was found to be able to chain the two elements together to perform KRACK. If older WNICs are also needed, as was the case with the KRACK testing scripts of [27], then the attacker may also have difficulties finding the right hardware. Then, the attacker is limited by the Wi-Fi range, and even more so by the attack which needs to move the client and AP to communicate with its malicious MITM device. Furthermore, the client must be vulnerable to the attack, which was shown to happen in rare cases.

On top of all these elements, other less technical attacks might be as efficient as KRACK in fulfilling the attacker's objective(s), as described in [2]. Typically, if the attacker aims for credentials, it could simply put its machine in a MITM position in case the data is not encrypted at a higher level of the OSI model (e.g., HTTPS or using a Virtual Private Network, or VPN). Of course, this wouldn't be as stealthy as the MC-MITM, but nonetheless efficient.

The seemingly difficult exploitation of KRACK is supported by the Exploit Prediction Scoring System (EPSS) score, which estimates how likely a vulnerability is to be exploited in the next 30 days [51]. Its scale goes from 0 to 100%, and the estimated EPSS score for CVE-2017-13077 was 0.16% in March of 2023, according to [52]. Overall, all KRACK-related CVEs had a score below 3%. The CVSSv3 Exploitability subscore of CVE-2017-13077 also indicates that exploitation is unlikely to occur, as it was set at 1.6 (out of 10) by NIST. The two scores suggest that KRACK is difficult to exploit. Although affecting the core of the WPA/WPA2's security, it seems that vendors and manufacturers patched the most problematic vulnerabilities correctly. As [2, p. 17] also notes, most of the attack techniques against Wi-Fi in 2018 were solved by moving to WPA3, including KRACK.

Still, the results of this analysis also show that more "minor" issues might remain unpatched or uncorrected in modern devices and OSs, such as accepting replayed broadcast frames or

ignoring the RSC reset performed for a GTK renewal. As WPA/WPA2 is still widely used, the presented results show that these issues should be addressed.

Closing this chapter, the assessed limits of the selected methods (see [Methods](#), p. 40) are discussed to present ideas for future work. It was indeed discovered that, given the time that passed since KRACK's discovery, very few devices were found to be reinstalling their keys (PTK, GTK), and thus reusing cryptographic nonces. Then, the representativity of the sampling was considered to be reasonably diverse. The selection process could have targeted Android/Linux-based devices more, as they were the main ones thought to be vulnerable to KRACK. Nevertheless, it was found that some Apple devices also had issues with some tests, which wouldn't have been discovered if only Android/Linux-based devices were selected.

The number of tested devices could be improved with more resources at disposal, and scripts to automate running the tests could be written to speed up the data collection process. Since connecting to a Wi-Fi network needs the manual approval of the device owner most of the time, a fully automated testing process is considered difficult to achieve. Further investigation into this option would be beneficial to similar studies aiming at detecting devices vulnerable to Wi-Fi attacks. Also, finding and sampling 802.11r APs to perform Vanhoef's test on them would also be an interesting continuation of this thesis. At a more general level, it would be interesting to develop a tool which could screen Wi-Fi-enabled devices for various Wi-Fi-related vulnerabilities, proving its result through traffic captures.



## 9 Conclusion

KRACK is a complex set of vulnerabilities affecting WPA/WPA2, two Wi-Fi security protocols still widely used in the world. It has been six years since its release, yet researchers have shown little attention to it, except for citing the discovered vulnerabilities as a potential type of attack against Wi-Fi. As shown, KRACK has been well patched over the years and it seems that WPA2 has still some years left before being fully replaced by WPA3, the current Wi-Fi security standard.

This thesis aimed to explain KRACK as thoroughly as possible. The contributions of this study was to detail the steps to install a testing environment, to review Vanhoef's testing script, to showcase how to run the tests against devices, as well as presenting traffic captures proving vulnerable behaviour. A convenience sampling was performed, collecting test results for 29 devices. For each device, a set of 7 tests based on Vanhoef's script was performed, totalling 203 outcomes. Only 2 older devices were identified as being vulnerable to the main KRACK vulnerabilities (CVE-2017-13077 and CVE-2017-13078). The devices reinstalled the PTK and GTK after retransmission of EAPOL Message 3, leading to nonce reuse. For this, variants of the simple retransmission of EAPOL Message 3 were also tested, with limited success. Also, 3 devices suffered from CVE-2017-13080, meaning that they reinstalled the GTK and reinitialise the related counter during the group key handshake. The script commands examining the reinitialisation of the RSC to a higher value were not efficient in revealing if devices were vulnerable to the tested malfunction. This analysis also discovered a logical bug which prevented one of Vanhoef's tests from running correctly, although this was not the reason why related results were unreliable. On top of device testing, this thesis provided Wireshark captures performed during the tests, deep diving into the traffic and related behaviour of examined devices. This allows the reader to get insights on the exchanged frames when KRACK is performed against a Wi-Fi-enabled device.

For future studies, developing scripts to automate the testing process could be written to speed up data collection, resulting in more representative analyses. Finally, a novel and broader research project could be the development of a general vulnerability detection system for Wi-Fi systems, which would screen devices with various test and proving the vulnerable behaviour through traffic capture.

## References

- [1] S. P. Bonacci and R. Wood, ‘Wireless network data traffic: worldwide trends and forecasts 2021–2026’, *Analysys Mason*. Accessed: May 07, 2023. [Online]. Available: [https://www.analysysmason.com/content-tassets/63ab4d5ecf364c60a558dcca49ee801f/analysys\\_mason\\_wireless\\_traffic\\_forecast\\_oct2021\\_samples\\_rdnt0.pdf](https://www.analysysmason.com/content-tassets/63ab4d5ecf364c60a558dcca49ee801f/analysys_mason_wireless_traffic_forecast_oct2021_samples_rdnt0.pdf)
- [2] C. Kohlios and T. Hayajneh, ‘A Comprehensive Attack Flow Model and Security Analysis for Wi-Fi and WPA3’, *Electronics*, vol. 7, no. 11, p. 284, Oct. 2018, doi: 10.3390/electronics7110284.
- [3] R. Katz, J. Jung, and F. Callorda, ‘The Economic Value of Wi-Fi A Global View 2021-2025’, Accessed: Mar. 19, 2023. [Online]. Available: [https://www.wi-fi.org/download.php?file=/sites/default/files/private/The\\_Economic\\_Value\\_of\\_Wi-Fi-A\\_Global\\_View\\_2021-2025\\_202109.pdf](https://www.wi-fi.org/download.php?file=/sites/default/files/private/The_Economic_Value_of_Wi-Fi-A_Global_View_2021-2025_202109.pdf)
- [4] E. Ferro and F. Potorti, ‘Bluetooth and wi-fi wireless protocols: a survey and a comparison’, *IEEE Wireless Commun.*, vol. 12, no. 1, pp. 12–26, Feb. 2005, doi: 10.1109/MWC.2005.1404569.
- [5] M. Vanhoef and F. Piessens, ‘Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2’, in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, Dallas Texas USA: ACM, Oct. 2017, pp. 1313–1328. doi: 10.1145/3133956.3134027.
- [6] P. Taylor, ‘Global mobile OS market share 2023’, *Statista*. <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/> (accessed Jul. 22, 2023).
- [7] ‘WiGLE Stats’. <https://wigle.net/stats#geostats> (accessed Jul. 22, 2023).
- [8] ‘IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements’. 2004. doi: 10.1109/IEEESTD.2004.94585.
- [9] M. Vanhoef and E. Ronnen, ‘Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd’, in *Proceedings of the IEEE Symposium on Security and Privacy, 2020- May*, 517–533, May 2020, pp. 517–533. doi: 10.1109/SP40000.2020.00031.

- [10] ‘Chapter 2. Overview of 802.11 Networks’, in *802.11 Wireless Networks: the definitive guide; [creating and administering Wireless Networks]*, 1. ed.in Creating and administering wireless networks. Beijing Köln: O’Reilly, 2002, pp. 20–34.
- [11] ‘Chapter 3. The 802.11 MAC’, in *802.11 Wireless Networks: the definitive guide; [creating and administering Wireless Networks]*, 1. ed.in Creating and administering wireless networks. Beijing Köln: O’Reilly, 2002, pp. 35–61.
- [12] ‘Chapter 4. 802.11 Framing in Detail’, in *802.11 Wireless Networks: the definitive guide; [creating and administering Wireless Networks]*, 1. ed.in Creating and administering wireless networks. Beijing Köln: O’Reilly, 2002, pp. 62–94.
- [13] ‘Standard 802.11-2007, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Subsection 8.3-RSNA data confidentiality protocols’. 2007.
- [14] M. Thankappan, H. Rifà-Pous, and C. Garrigues, ‘Multi-Channel Man-in-the-Middle attacks against protected Wi-Fi networks: A state of the art review’, *Expert Systems with Applications*, vol. 210, p. 118401, Dec. 2022, doi: 10.1016/j.eswa.2022.118401.
- [15] ‘Chapter 7. 802.11i: Robust Security Networks, TKIP, and CCMP’, in *802.11 Wireless Networks: the definitive guide; [creating and administering Wireless Networks]*, 1. ed.in Creating and administering wireless networks. Beijing Köln: O’Reilly, 2002, pp. 184–203.
- [16] M. Vanhoef, ‘KRACK Attacks: Breaking WPA2’. <https://www.krackattacks.com/> (accessed Jul. 22, 2023).
- [17] S. Fluhrer, I. Mantin, and A. Shamir, ‘Weaknesses in the Key Scheduling Algorithm of RC4’, in *Selected Areas in Cryptography*, S. Vaudenay and A. M. Youssef, Eds., in Lecture Notes in Computer Science, vol. 2259. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1–24. doi: 10.1007/3-540-45537-X\_1.
- [18] A. Stubblefield, J. Ioannidis, and A. D. Rubin, ‘Using the Fluhrer, Mantin, and Shamir Attack to Break WEP’, *AT&T Labs Technical Report TD-4ZCPZZ*, 2001.
- [19] E. Tews, R.-P. Weinmann, and A. Pyshkin, ‘Breaking 104 Bit WEP in Less Than 60 Seconds’, in *Information Security Applications*, S. Kim, M. Yung, and H.-W. Lee, Eds., in Lecture Notes in Computer Science, vol. 4867. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 188–202. doi: 10.1007/978-3-540-77535-5\_14.
- [20] E. Tews and M. Beck, ‘Practical attacks against WEP and WPA’, in *Proceedings of the second ACM conference on Wireless network security - WiSec ’09*, Zurich, Switzerland: ACM Press, 2009, p. 79. doi: 10.1145/1514274.1514286.

- [21] S. Viehböck, ‘Brute forcing Wi-Fi Protected Setup’, *CERT Vulnerability Note VU 723755*, 2011.
- [22] C. He and J. C. Mitchell, ‘Analysis of the 802.11i 4-way handshake’, in *Proceedings of the 2004 ACM workshop on Wireless security - WiSe '04*, Philadelphia, PA, USA: ACM Press, 2004, p. 43. doi: 10.1145/1023646.1023655.
- [23] M. Cunche, ‘Comprendre les attaques KRACK’, no. 99 (HS), Nov. 2018. Accessed: Jun. 11, 2023. [Online]. Available: <https://connect.ed-diamond.com/GNU-Linux-Magazine/glmfhs-099/comprendre-les-attaques-krack>
- [24] M. Vanhoef, ‘Chromium Bug Tracker: WPA1/2 all-zero session key & key reinstallation attacks.’ 2017. Accessed: Aug. 29, 2017. [Online]. Available: <https://bugs.chromium.org/p/chromium/issues/detail?id=743276>
- [25] M. Vanhoef and F. Piessens, ‘Release the Kraken: New KRACKs in the 802.11 Standard’, in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto Canada: ACM, Oct. 2018, pp. 299–314. doi: 10.1145/3243734.3243807.
- [26] J. Ernekerová, ‘Analysis and Detection of KRACK Attack Against WiFi Infrastructure’, Master’s thesis, Czech Technical University in Prague, Faculty of Information Technology, 2019.
- [27] M. Vanhoef, ‘krackattacks-scripts’. Jul. 20, 2023. Accessed: Jul. 23, 2023. [Online]. Available: <https://github.com/vanhoefm/krackattacks-scripts>
- [28] M. Vanhoef, ‘Fragment and Forge: Breaking Wi-Fi Through Frame Aggregation and Fragmentation’, *30th USENIX Security Symposium*, 2021.
- [29] M. Čermák, Š. Svorenčík, R. Lipovský, and O. Kubovič, ‘KR00K - Serious vulnerability deep inside your wi-fi encryption’, *ESET White Paper*, 2020.
- [30] J. Freudenreich, J. Weidman, and J. Grossklags, ‘Responding to KRACK: Wi-Fi Security Awareness in Private Households’, in *Human Aspects of Information Security and Assurance*, N. Clarke and S. Furnell, Eds., in IFIP Advances in Information and Communication Technology, vol. 593. Cham: Springer International Publishing, 2020, pp. 233–243. doi: 10.1007/978-3-030-57404-8\_18.
- [31] N. Sombatruang, Y. Kadobayashi, M. A. Sasse, M. Baddeley, and D. Miyamoto, ‘The continued risks of unsecured public Wi-Fi and why users keep using it: Evidence from Japan’, in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, Belfast: IEEE, Aug. 2018, pp. 1–11. doi: 10.1109/PST.2018.8514208.

- [32] O. Nakhila, A. Attiah, Y. Jin, and C. Zou, ‘Parallel active dictionary attack on WPA2-PSK Wi-Fi networks’, in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, Tampa, FL: IEEE, Oct. 2015, pp. 665–670. doi: 10.1109/MILCOM.2015.7357520.
- [33] D. Schepers, A. Ranganathan, and M. Vanhoef, ‘Practical Side-Channel Attacks against WPA-TKIP’, in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, Auckland New Zealand: ACM, Jul. 2019, pp. 415–426. doi: 10.1145/3321705.3329832.
- [34] H. Musthyala and P. N. Reddy, ‘Hacking wireless network credentials by performing phishing attack using Python Scripting’, in *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India: IEEE, May 2021, pp. 248–253. doi: 10.1109/ICICCS51141.2021.9432155.
- [35] B. Pingle, A. Mairaj, and A. Y. Javaid, ‘Real-World Man-in-the-Middle (MITM) Attack Implementation Using Open Source Tools for Instructional Use’, in *2018 IEEE International Conference on Electro/Information Technology (EIT)*, Rochester, MI: IEEE, May 2018, pp. 0192–0197. doi: 10.1109/EIT.2018.8500082.
- [36] M. Vondráček, J. Pluskal, and O. Ryšavý, ‘Automated Man-in-the-Middle Attack Against Wi-Fi Networks’, *JDFSL*, 2018, doi: 10.15394/jdfsl.2018.1495.
- [37] M. Denis, C. Zena, and T. Hayajneh, ‘Penetration testing: Concepts, attack methods, and defense strategies’, in *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, Farmingdale, NY, USA: IEEE, Apr. 2016, pp. 1–6. doi: 10.1109/LISAT.2016.7494156.
- [38] M. E. Garbelini, C. Wang, and S. Chattopadhyay, ‘Greyhound: Directed Greybox Wi-Fi Fuzzing’, *IEEE Trans. Dependable and Secure Comput.*, vol. 19, no. 2, pp. 817–834, Mar. 2022, doi: 10.1109/TDSC.2020.3014624.
- [39] M. A. Abo-Soliman and M. A. Azer, ‘A study in WPA2 enterprise recent attacks’, in *2017 13th International Computer Engineering Conference (ICENCO)*, Cairo: IEEE, Dec. 2017, pp. 323–330. doi: 10.1109/ICENCO.2017.8289808.
- [40] A. Bartoli, E. Medvet, and F. Onesti, ‘Evil twins and WPA2 Enterprise: A coming security disaster?’, *Computers & Security*, vol. 74, pp. 1–11, May 2018, doi: 10.1016/j.cose.2017.12.011.

- [41] Z. C. Schreuders and A. M. Bhat, ‘Not All ISPs Equally Secure Home Users - An Empirical Study Comparing Wi-Fi Security Provided by UK ISPs’, in *Proceedings of the 10th International Conference on Security and Cryptography*, Reykjavík, Iceland: SCITE-PRESS - Science and Technology Publications, 2013, pp. 568–573. doi: 10.5220/0004600405680573.
- [42] V. O. Etta, A. Sari, A. L. Imoize, P. K. Shukla, and M. Alhassan, ‘Assessment and Test-case Study of Wi-Fi Security through the Wardriving Technique’, *Mobile Information Systems*, vol. 2022, pp. 1–21, Jun. 2022, doi: 10.1155/2022/7936236.
- [43] R. Kalniņš, J. Puriņš, and G. Alksnis, ‘Security Evaluation of Wireless Network Access Points’, *Applied Computer Systems*, vol. 21, no. 1, pp. 38–45, May 2017, doi: 10.1515/acss-2017-0005.
- [44] A. Sebbar, Se. Boulahya, G. Mezzour, and M. Boulmalf, ‘An empirical study of WIFI security and performance in Morocco - wardriving in Rabat’, in *2016 International Conference on Electrical and Information Technologies (ICEIT)*, Tangiers, Morocco: IEEE, May 2016, pp. 362–367. doi: 10.1109/EITech.2016.7519621.
- [45] I. Hossain, M. M. Hasan, S. Faisal Hasan, and Md. R. Karim, ‘A study of security awareness in Dhaka city using a portable WiFi pentesting device’, in *2019 2nd International Conference on Innovation in Engineering and Technology (ICIET)*, Dhaka, Bangladesh: IEEE, Dec. 2019, pp. 1–6. doi: 10.1109/ICIET48527.2019.9290589.
- [46] D. Delija, Z. Petrovic, G. Sirovatka, and M. Zagar, ‘An Analysis of Wireless Network Security Test Results provided by Raspberry Pi Devices on Kali Linux’, in *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, Opatija, Croatia: IEEE, Sep. 2021, pp. 1219–1223. doi: 10.23919/MIPRO52101.2021.9596954.
- [47] M. Vanhoef, ‘Multi-Channel Machine-in-the-Middle’. Jul. 05, 2023. Accessed: Jul. 23, 2023. [Online]. Available: <https://github.com/vanhoefm/mc-mitm>
- [48] M. Vanhoef, ‘krackattacks-poc-zerokey’. 2018. Accessed: Jul. 23, 2023. [Online]. Available: <https://github.com/vanhoefm/krackattacks-poc-zerokey/blob/research/krack-attack/krack-all-zero-tk.py>
- [49] L. Couto, ‘fork of krackattacks-scripts’. Nov. 19, 2021. Accessed: Jul. 23, 2023. [Online]. Available: <https://github.com/lucascouto/krackattacks-scripts>

- [50] M. Thankappan, ‘Multi Channel Man-in-the-Middle Attacks Against Protected Wi-Fi Networks’. Mar. 31, 2022. Accessed: Jul. 23, 2023. [Online]. Available: <https://github.com/maneshthankappan/Multi-Channel-Man-in-the-Middle-Attacks-Against-Protected-Wi-Fi-Networks-By-Improved-Variant>
- [51] ‘Exploit Prediction Scoring System (EPSS)’, *FIRST — Forum of Incident Response and Security Teams*. <https://www.first.org/epss> (accessed Aug. 04, 2023).
- [52] ‘Vulnerability Details: CVE-2017-13077’. <https://www.cvedetails.com/cve/CVE-2017-13077/> (accessed Aug. 04, 2023).

## Appendix

### I. Filters used for Wireshark Capture Figures

**Figure 20:** Detailed look at the CCMP Header of frame no. 168 (p. 56)

Wireshark filter used:

`(wlan.sa == c0:1c:30:2f:f6:f0) || (wlan.sa == 22:61:71:23:de:e6)`

Explanation: It filters for the Source address (“sa”) respectively using the AP’s and the device no. 12’s MAC address. The logical OR symbol is represented by “||” (which is used here), while the AND symbol is represented by “&&”. For each Wireshark capture (i.e., figure), the filters used for the display are defined.

**Figure 21:** *tptk-rand* variant of KRACK and Disassociation as a response by device no. 12 (Test 4) (p. 58)

Wireshark filter used:

`(wlan.sa == c0:1c:30:2f:f6:f0) || (wlan.sa == 22:61:71:23:de:e6)`

**Figure 22:** Wrong behaviour by *hostapd* for Test 5 (device no. 12) (p. 58)

Wireshark filter used:

`eapol`

Explanation: As suggested by its name, this filter only displays EAPOL messages.

**Figure 24:** RSC value in EAPOL group message 1 is not set by the client (device no. 12, Test 7) (p. 59)

Wireshark filter used:

`eapol`

**Figure 25:** Test 1 on device no. 12 (Fairphone FP4) with the first KRACK attempt highlighted (p. 60)

Wireshark filter used:

`eapol`

**Figure 26:** No PTK reinstallation, as no PTK-IV was reused on device no. 12 (with Test 1) (p. 61)



Wireshark filter used:

`(wlan.wep.key == 0) && (wlan.ta == 22:61:71:23:de:e6)`

Explanation: It filters for the Transmitter address (“ta”) of the device no. 12’s MAC address (and not the Source address), as well as frames encrypted using the TK. The reason for selecting “ta” instead of “sa” is to ignore the broadcast frames sent by the client and avoid confusion.

Indeed, these “broadcast” frames are, in fact, a combination of “unicast (by client) + broadcast (by AP)” frames, as stated before. Using the “ta” filter skips these frames.

**Figure 27:** No replies from device no. 12 to replayed ARP requests (Test 1) (p. 61)

Wireshark filter used:

`arp`

Explanation: Only ARP frame will be displayed.

**Figure 29:** IV reuse by device no. 3 (Test 1) (p. 62)

Wireshark filter used:

`((wlan.sa == c0:1c:30:2f:f6:f0) || (wlan.sa == f4:0b:93:66:fc:9f)) && (wlan.fc.type == 2)`

Explanation: The last parameter filters for Data frames to give a clearer but still relevant view to the reader.

**Figure 31:** Absence of replies to replayed ARP requests for device no. 3 (Test 1) (p. 63)

Wireshark filter used:

`arp`

**Figure 33:** Multiple IV reuse due to repeated PTK reinstallations on device no. 3 (Test 3) (p. 64)

Wireshark filter used:

`eapol`

**Figure 34:** As for the Fairphone, the RSC value is not incremented for the Blackberry (Test 7) (p. 64)

Wireshark filter used:

`arp`

**Figure 37:** Constant RSC reset to 0 by Redmi Note 9 during the group key handshake (Test 7) (p. 65)

Wireshark filter used:

*eapol*

**Figure 38:** RSC is also not incremented in the group key handshake for Asus Zenfone 6 (Test 7) (p. 65)

Wireshark filter used:

*eapol*

**Figure 41:** IV reuse highlighted due to PTK reinstallation on Samsung S5 (Test 1) (p. 66)

Wireshark filter used:

*((wlan.wep.key == 0 || eapol) && wlan.sa == 60:af:6d:f9:69:c0) || (eapol && wlan.sa == c0:1c:30:2f:f6:f0)*

Explanation: To summarise this filter, TK-encrypted or EAPOL frames sent by the client are displayed, as well as EAPOL frames sent by the AP.

**Figure 42:** Device no. 4 repeatedly replies to ARP requests, indicating GTK reinstallation (Test 1) (p. 67)

Wireshark filter used:

*arp*

**Figure 44:** Multiple ARP responses to replayed ARP requests by device no. 4 (Test 5) (p. 67)

Wireshark filter used:

*arp || ((wlan.ta == c0:1c:30:2f:f6:f0) && (wlan.wep.key == 1))*

Explanation: The filter here expands the view which just showed ARP frames: it also includes GTK-encrypted frames for which the Transmitter address is the AP's.

**Figure 45:** A replayed ARP request accepted by device no. 1 (Test 2) (p. 68)

Wireshark filter used:

*((arp.src.hw\_mac == b4:ae:2b:2c:b7:e9) || (arp.src.hw\_mac == c0:1c:30:2f:f6:f0))*

Explanation: It filters for ARP requests sent by these 2 MAC addresses (which are, respectively, the Surface's (device no. 1) and the AP's MAC address.

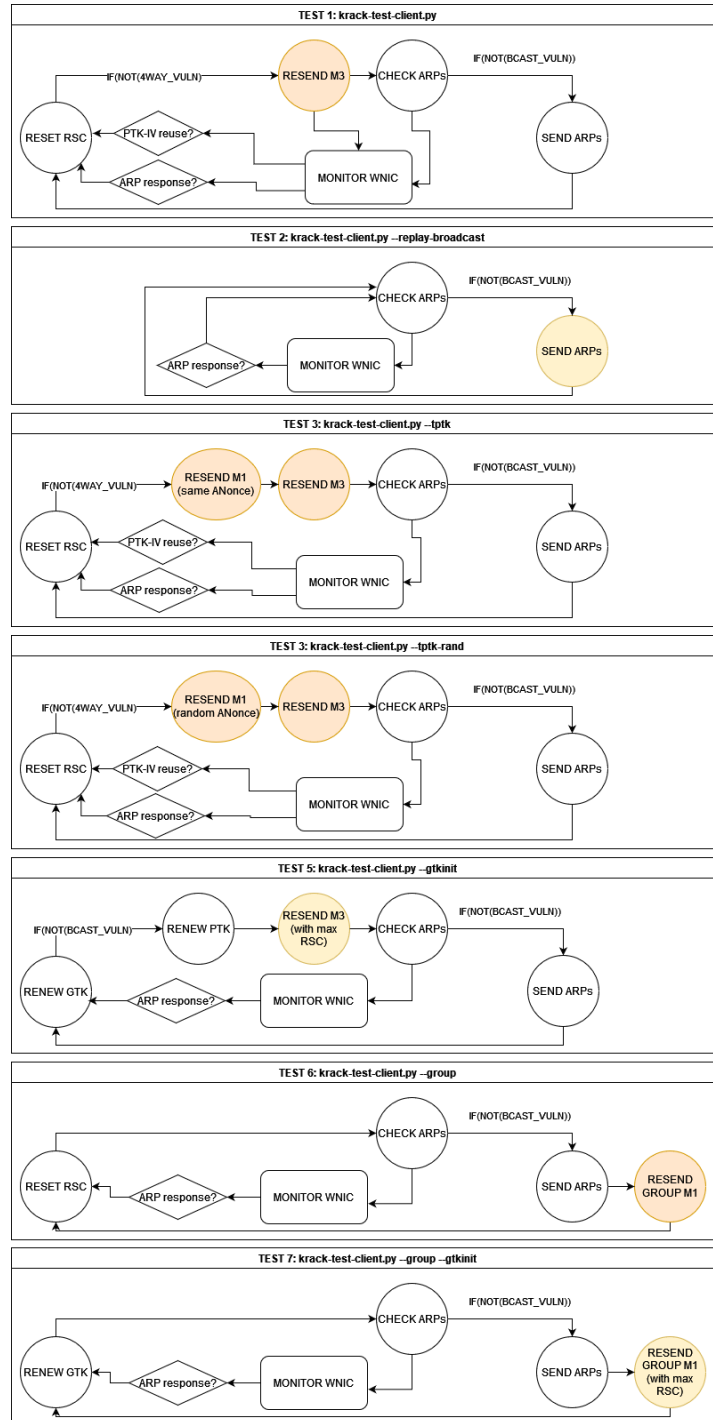
**Figure 46:** Expanded view of traffic with the Surface 3 device to follow GTK-IVs (Test 2) (p. 68)

Wireshark filter used:

*arp || ((wlan.ta == c0:1c:30:2f:f6:f0) && (wlan.wep.key == 1))*

## II. Step-by-step Process of KRACK Tests

This additional section aims at summarising the steps and checks each test goes through<sup>25</sup>. Attempts targeting the 3 KRACK CVEs which were tested are highlighted in orange, while the yellow forms highlight the cause of the malfunction.



<sup>25</sup> “BCAST\_VULN” checks if replayed ARPs received replies. “4WAY\_VULN” checks if PTK-IVs were reused.

### **III. License**

#### **Non-exclusive licence to reproduce thesis and make thesis public**

**I, Brice Seiler,**

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

**Vulnerability of Wi-Fi-enabled Devices to KRACK Attacks – A Case Study,**

supervised by Danielle Morgan,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **11.08.2023**