Release the Kraken: New KRACKs in the 802.11 Standard

Mathy Vanhoef
imec-DistriNet, KU Leuven
Mathy.Vanhoef@cs.kuleuven.be

Frank Piessens
imec-DistriNet, KU Leuven
Frank.Piessens@cs.kuleuven.be

ABSTRACT

We improve key reinstallation attacks (KRACKs) against 802.11 by generalizing known attacks, systematically analyzing all handshakes, bypassing 802.11's official countermeasure, auditing (flawed) patches, and enhancing attacks using implementation-specific bugs.

Last year it was shown that several handshakes in the 802.11 standard were vulnerable to key reinstallation attacks. These attacks manipulate handshake messages to reinstall an already-in-use key, leading to both nonce reuse and replay attacks. We extend this work in several directions. First, we generalize attacks against the 4-way handshake so they no longer rely on hard-to-win race conditions, and we employ a more practical method to obtain the required man-in-the-middle (MitM) position. Second, we systematically investigate the 802.11 standard for key reinstallation vulnerabilities, and show that the Fast Initial Link Setup (FILS) and Tunneled direct-link setup PeerKey (TPK) handshakes are also vulnerable to key reinstallations. These handshakes increase roaming speed, and enable direct connectivity between clients, respectively. Third, we abuse Wireless Network Management (WNM) power-save features to trigger reinstallations of the group key. Moreover, we bypass (and improve) the official countermeasure of 802.11. In particular, group key reinstallations were still possible by combining EAPOL-Key and WNM-Sleep frames. We also found implementation-specific flaws that facilitate key reinstallations. For example, some devices reuse the ANonce and SNonce in the 4-way handshake, accept replayed message 4's, or improperly install the group key.

We conclude that preventing key reinstallations is harder than expected, and believe that (formally) modeling 802.11 would help to better secure both implementations and the standard itself.

CCS CONCEPTS

• **Security and privacy → Security protocols**; **Mobile and wireless security**; • **Networks → Security protocols**;

KEYWORDS

key reinstallation attack; KRACK; WPA2; 802.11; security protocols

ACM Reference Format:
Mathy Vanhoef and Frank Piessens. 2018. Release the Kraken: New KRACKs in the 802.11 Standard. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3243734.3243807

CCS '18, October 15–19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243807

1 INTRODUCTION

Last year, Vanhoef and Piessens showed that WPA2 is vulnerable to key reinstallation attacks [50]. What made their attack surprising, apart from taking more than a decade to discover, is that the core components of WPA2 were formally proven secure. That is, both the 4-way handshake and the (AES-)CCMP encryption protocol had security proofs [19, 25]. However, the manner in which these two components interacted made it possible to reinstall an already-in-use key. This resets the key's associated parameters such as transmit nonces and receive replay counters, making it possible to decrypt, replay, and potentially forge packets. Because this was a flaw in the standard, all WPA2 implementations were affected by some variant of the attack [50]. We build upon this work, and extend it in several directions. Summarized, we demonstrate that key reinstallation attacks can be further improved, and unfortunately are harder to prevent than initially assumed.

We first increase the practicality of key reinstallation attacks against the 4-way handshake. Previously, device-specific and hard-to-win race conditions had to be used to exploit the 4-way handshake of Android, macOS, and OpenBSD [50]. This was necessary, because otherwise these platforms would not accept the plaintext handshake message that triggers the key reinstallation. Moreover, it was not possible to attack Wi-Fi drivers of OpenBSD that use software encryption. We overcome these limitations by generating an encrypted (instead of plaintext) handshake message that triggers the key reinstallation. As a result, an adversary no longer has to rely on hard-to-win race conditions to exploit vulnerable implementations of the 4-way handshake. The encrypted handshake message can also be used to attack OpenBSD irregardless of the used Wi-Fi driver. Apart from making it easier to trigger key reinstallations, we will also discuss an easier method to establish the required man-in-the-middle (MitM) position.

We continue by systematically investigating all 802.11 features that may be affected by key reinstallation attacks. In particular, we inspected the 802.11 standard for handshakes that negotiate and install keys, and we searched for frames that transport keys. Then we looked for non-standard functionality that installs keys, by auditing the open source code of wpa_supplicant, hostapd, and iwd. Finally, we also checked whether vendors correctly patched known key reinstallation vulnerabilities. This systematic analysis revealed several new vulnerabilities.

Our first discovery is that both the FILS and TPK handshake are also vulnerable to key reinstallations. The FILS handshake can establish a secure link and Internet connection in only 100 ms, and is expected to be widely adopted in highly mobile environments. The TPK handshake is mainly used to stream data from a device to a smart TV [56], and establishes a direct tunnel between two clients. We demonstrate how an adversary can trigger key reinstallations against these handshakes, making it possible to decrypt, replay, and possibly forge frames.

When searching for frames that transports keys, we found that certain WNM-Sleep frames may transport the (integrity) group key. We demonstrate that manipulating these frames allows an adversary to trigger a reinstallation of the (integrity) group key. Building on top of this, we also show how to break the official countermeasure of 802.11 that is supposed to prevent key reinstallation attacks. This is done by exploiting interactions between EAPOL-Key frames and WNM-Sleep frames. More precisely, our attack first lets the victim install a new (integrity) group key, so it can then be tricked into reinstalling an older (integrity) group key. Apart from this, we also encountered implementation-specific vulnerabilities where the group key is improperly installed. For example, we found that certain devices always install the group key with a zero replay counter, instead of the given replay counter. More troublesome, we even discovered some devices that always accept replayed broadcast and multicast frames. All combined, this shows that securely handling and installing group keys is a non-trivial task.

In the last part of the paper, we present several implementation-specific key reinstallation vulnerabilities that we discovered when inspecting patches and open source code. First, we discuss implementations that reuse the SNonce or ANonce when refreshing the session key using the 4-way handshake. We demonstrate how this can be abused to trigger key reinstallations. A second noteworthy vulnerability is that some Access Points (APs) accept replayed message 4's of the 4-way handshake. An adversary can abuse this to trivially trigger key reinstallations, without having to obtain a MitM. In turn it becomes possible to decrypt, replay, and possibly forge frames. Finally, we show how to perform key reinstallation attacks against Android 7.0 and higher without relying on device-specific race conditions. These results show that preventing key reinstallations can be more tedious in practice than expected.

To summarize, our main contributions are:

- We show how to attack the 4-way handshake without relying on hard-to-win race conditions, and use a method to more easily obtain the required multi-channel MitM.
- We systematically analyze all 802.11 features that negotiate or manage keys, and discover that the FILS and TPK handshake are also vulnerable to key reinstallations.
- We show that the updated 802.11 standard is still vulnerable to reinstallations of the group key, and present implementation flaws that affect the security of group-addressed frames.
- We analyze security patches of vendors, and discover several implementation-specific key (re)installation vulnerabilities.

The remainder of this paper is organized as follows. Section 2 introduces the 802.11 standard and key reinstallation attacks. In Section 3 we generalize attacks against the 4-way handshake, and make it easier to obtain a MitM. We attack the FILS handshake in Section 4 and the TPK handshake in Section 5. In Section 6 we show how WNM frames can be used to reinstall the group key, and Section 7 discusses implementation-specific vulnerabilities. Finally, we discuss related work in Section 8 and conclude in Section 9.

2 BACKGROUND

In this section we introduce relevant parts of the 802.11 standard, and explain the core idea behind a key reinstallation attack [20, 50].

FC	addr1	addr2	addr3	KeyID & PN	Data

Figure 1: Simplified 802.11 frame with a WPA2 header.

2.1 The 802.11 Standard

A simplified layout of a 802.11 frame is shown in Figure 1. First, the Frame Control (FC) field contains several bit-flags, with the most important one for us being the Power Management (PM) flag. A station sets the PM flag in an (empty) data frame to indicate it is entering sleep mode, meaning it will no longer be able to receive frames. For ease of readability, we will refer to the PM flag as the sleep flag. The AP will buffer frames for clients that are asleep. To exit sleep mode, the station sends an (empty) frame where the sleep flag is not set. More advanced power management techniques will be further discussed (and abused) in Section 6.

The next three fields contain the address of the receiver (addr1), the address of the sender (addr2), and the address of the final destination (addr3). For example, when a client sends an outbound IP packet, addr1 equals the MAC address of the AP, addr2 that of the client itself, and addr3 that of the router. A station, i.e., a client or AP, can also combine several frames into a single A-MSDU frame. This avoids the overhead of sending multiple small frames. The content of each frame, along with its source and destination address, is stored in the data field of the A-MSDU frame. Interestingly, against version 4.8 and below of the Linux kernel, A-MSDU frames can be abused to spoof the sender and destination addresses of frames [5].

When a frame is encrypted, its plaintext header includes the KeyID and Packet Number (PN) field. The PN field stores the replay counter used by the encryption algorithm (see Section 2.4), and the KeyID identifies which key was used to protect the frame. Modern networks mainly use the KeyID to transparently update the group key. That is, a newly generated group key uses a different KeyID than the one currently in use. This allows an AP to (gradually) distribute the new group key, while simultaneously sending group-addressed frames using the old key.

Before a client can send data frames, it first needs to authenticate and associate with the AP. This is illustrated in the first stage of Figure 2. At this stage, real authentication only takes place when using the FT, SAE, or FILS handshake [20]. Most WPA2 networks do not yet support these handshakes, and instead use Open System authentication. This means there is no authentication at this stage, and actual authentication is offloaded to the 4-way handshake.

After authentication, the client (re)associates with the AP. In this process the client informs the AP which features it supports and wants to use (e.g. its supported bitrates). Additionally, if encryption is used, the (re)association frame contains the cipher suite that the client wishes to use. The AP replies with a (re)association response that indicates whether the association was successful, i.e., whether the requested features and cipher suite are supported.

Finally, the 802.11 standard supports Quality of Service (QoS) features to prioritize certain traffic. In particular, the standard defines 16 different priority channels [20, §5.1.1.3], where each channel is identified by a 4-bit Traffic Identifier (TID). Interestingly, when encryption is used, each QoS priority channel uses its own receive replay counter (under the same session key). However, a

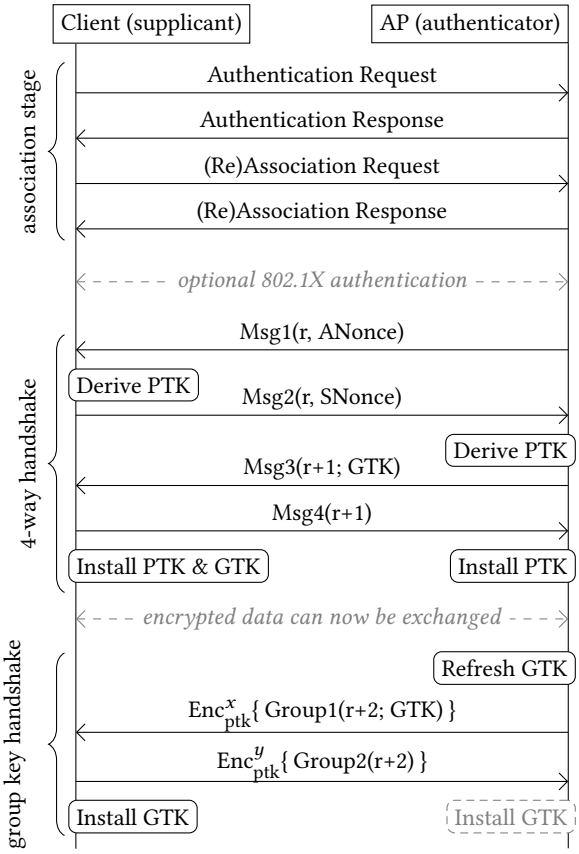

Figure 2: Messages exchanged when a client connects with an AP, performs the 4-way handshake, and periodically executes the group key handshake [50].

single transmit replay counter is shared by all QoS channels. This design allows the physical-layer component of the Wi-Fi chip to reorder outgoing frames based on their priority. We will abuse these priority-dependent receive replay counters in Section 3.3 and 5.4.

2.2 Protected Wi-Fi Networks

All modern protected Wi-Fi networks rely on the 802.11i amendment, which defines both the 4-way handshake and two encryption protocols [22]. However, due to the slow standardization of this amendment, the Wi-Fi Alliance already started certifying devices based on a draft of 802.11i under the Wi-Fi Protected Access (WPA) program. Once 802.11i was finished, the WPA2 certification program was created. As a result, WPA and WPA2 are very similar. The main difference is that WPA2 mandates support for the more secure (AES-)CCMP encryption protocol, and optionally allows the (WPA-)TKIP encryption protocol, while the reverse is true for WPA.

As a reaction to the key reinstallation attacks against WPA2, the Wi-Fi Alliance recently released the WPA3 certification program [57]. This certification mandates support of the Simultaneous Authentication of Equals (SAE) handshake. In contrast to the 4-way handshake of WPA2, the SAE handshake provides forward secrecy, and is resistant to dictionary attacks.

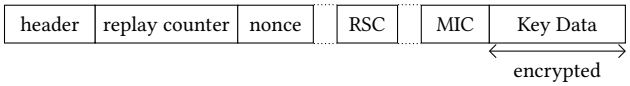

Figure 3: Simplified layout of an EAPOL-Key frame.

2.3 The 4-way Handshake

The 4-way handshake provides both mutual authentication and session key generation. Authentication is based on a shared secret called the Pairwise Master Key (PMK), and from this a fresh session key called the Pairwise Transient Key (PTK) is derived. In this handshake the client is called the supplicant, and the AP is called the authenticator. Concretely, the PTK is a combination of the PMK, the Authenticator Nonce (ANonce), the Supplicant Nonce (SNonce), and the MAC address of both the supplicant and client. In turn, the PMK is either derived from a pre-shared password in a personal network, or negotiated using 802.1X in an enterprise network.

Every message in the 4-way handshake is defined using EAPOL-Key frames. Their most important fields are shown in Figure 3. The header contains eight flags which describe properties of the frame. For instance, the Encrypted flag denotes whether the key data field is encrypted or not, and the MIC flag denotes whether the frame is authenticated using a Message Integrity Check (MIC). Note that for every message in the handshake, a unique combination of flags is set. The replay counter is used to detect replayed messages. That is, the AP increments the replay counter for every handshake message it sends. When the client replies to the AP, it uses the same replay counter that it previously received from the AP. The nonce field transports either the random ANonce or SNonce of the AP or client, respectively. When the frame also transports a (integrity) group key (GTK), it is saved in the key data field, which is encrypted using the PTK. The Receive Sequence Counter (RSC) contains the current replay counter of the transported group key. In contrast, the current replay counter of the integrity group key is saved in the key data field. We will use the notation GTK_i^z to represent a group key with replay counter z and key id i. When the value of either z or i is not important, we will not include them in the notation.

Figure 2 illustrates the messages exchanged in the 4-way handshake. Similar as in [50], we use the notation

MsgN(r, nonce; GTK)

to represent message n of the 4-way handshake, which uses the replay counter r, and the given nonce (if present). All parameters after the semicolon are stored in the key data field. Simplified, the first two messages of the 4-way handshake are used to exchange nonces, and the last two messages are used to confirm that both parties derived the same session key. Note that message 3 transports the group key to the client. The authenticity of every message, except Msg1, is protected with a MIC that is calculated using the KCK subkey of the PTK. After the 4-way handshake, the AP can use the group key handshake to distribute (periodically renewed) group keys to all associated clients (see Figure 2).

The PTK can be refreshed by performing a new 4-way handshake after the initial one. In this new handshake, messages are encrypted using the previously negotiated PTK. More importantly, the client and AP generate a fresh SNonce and ANonce, respectively. By refreshing these nonces, a new PTK will be negotiated.

2.4 Encryption Protocols

The 802.11 standard defines four encryption protocols when excluding the broken WEP protocol. The first three are (WPA-)TKIP, (AES-)CCMP, and GCMP, and they are used to protect data frames and certain unicast management frames. All three behave like stream ciphers, meaning keystream is generated and XORed with the plaintext data. The generated keystream depends on the TK subkey of the PTK, and on a 48-bit packet number. This packet number, commonly called a nonce, is incremented by one for every transmitted packet, initialized to zero or one depending on the specific protocol, and used as a replay counter by the receiver.

The fourth encryption protocol is the Broadcast/multicast Integrity Protocol (BIP), and is designed to authenticate (but not encrypt) group-addressed robust management frames. It uses an integrity group key for protection, employs a 48-bit packet number to detect replays, and calculates an authenticity tag using AES-CMAC.

To denote that a frame is encrypted and authenticated we use the following notation:

$$\text{Enc}_k^n\{\cdot\}$$

Here n denotes the packet number being used (and thus also the replay counter). The parameter k denotes the used key, which is the session key (PTK) for unicast traffic, and the group key (GTK) for multicast and broadcast traffic. Finally, the two notations

[header,] Data(payload)
[header,] GroupData(payload)

represent an ordinary unicast or broadcast data frame, respectively, with the given payload and 802.11 header. The optional header represents the unauthenticated plaintext header of the 802.11 frame. It may contain the sleep parameter, which denotes that the Power Management flag is set in the FC field (recall Section 2.1). When the sleep flag is not present, the corresponding PM flag is not set.

In case of a key reinstallation, the transmit nonce is reset to zero and subsequently reused. This implies keystream is reused during encryption, making it possible to decrypt frames. Moreover, against (WPA-)TKIP and GCMP, the adversary can recover the authentication key if a nonce is ever reused [26, 43]. This means it becomes possible to forge frames as well. Finally, a key reinstallation also resets the receive replay counter, meaning an adversary can replay old frames towards the victim.

2.5 Key Reinstallation Attacks

More than a decade after the creation of WPA2, Vanhoef and Piessens discovered that it is vulnerable to key reinstallation attacks [50]. Surprisingly, the attack is easy to understand once pointed out. The main idea is that an adversary abuses retransmissions of handshake messages to trick a victim into reinstalling a key. This causes the key's associated parameters, such as the incremental transmit nonce and receive replay counter, to be reset. As mentioned in Section 2.4, this allows an adversary to decrypt, replay, and possibly forge frames.

Figure 4 shows how to perform a key reinstallation attack against the 4-way handshake. First, the adversary obtains a multi-channel man-in-the-middle (MitM) position [50]. This position does not enable decryption of frames, but only the ability to reliably block and delay messages sent between the client and AP [47]. The MitM

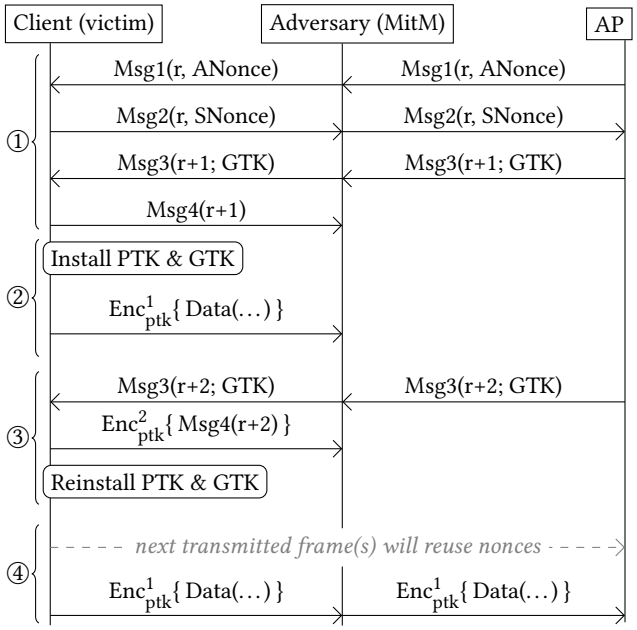

Figure 4: Key reinstallation attack against the 4-way handshake, when the client (victim) still accepts plaintext retransmissions of message 3if a PTK is installed [50].

works by cloning the AP on a different channel, forcing the victim to connect to the AP on the rogue channel, and then forwarding frames between both channels. Once in this position, the adversary forwards the first three messages of the 4-way handshake without modification, but does not forward message 4 (see stage 1 of Figure 4). Nevertheless, the client considers the handshake to be complete, meaning it will install the negotiated session key (PTK). As a result, any data frames that the client now transmits are encrypted at the link layer (see stage 2 in Figure 4). However, because the AP did not receive message 4, it has not yet completed the handshake. To remedy this, the AP will retransmit a new message 3 using an increased replay counter of $r + 2$. When the client receives this retransmitted message 3, it will reply using a new message 4. Additionally, it will reinstall the PTK, and thereby reset the transmit nonce and receive replay counter. This is catastrophic, as it causes nonce reuse when sending the next data frame (see stage 4 in Figure 4). Additionally, it becomes possible to replay frames towards the victim, since the receive replay counter was also reset.

A similar attack was presented against the PeerKey, group key, and Fast BSS Transition (FT) handshake. For more details we refer the reader to [50]. In this paper, we also demonstrate that the FILS and TPK handshakes are vulnerable to key reinstallations.

After the disclosure of the attacks, the 802.11 standard was modified in an attempt to prevent all key reinstallation vulnerabilities [17]. Interestingly, this modification consisted of just two new lines, with the main one being [17]:

> "When the Key, Address, Key Type, and Key ID parameters identify an existing key, the MAC shall not change the current transmitter TSC/PN/IPN counter or the receiver replay counter values associated with that key."

In other words, when an already-in-use key is being reinstalled, the standard now states that the associated transmit nonce and receive replay counter should not be reset. Unfortunately, in Section 6.3 we will demonstrate that this defense does not prevent all attacks.

3 IMPROVED 4-WAY HANDSHAKE ATTACKS

In this section we present an attack against vulnerable implementations of the 4-way handshake that does not rely on hard-to-win race conditions. Additionally, we show how to easily obtain a multi-channel MitM and increase the impact of key reinstallations.

3.1 Existing Attacks and their Limitations

The original key reinstallation attack against the 4-way handshake of Android, macOS, and OpenBSD, relied on hard-to-win race conditions to trigger the key reinstallation [50]. This was necessary because, by default, these implementations no longer accept plaintext handshake messages after installing the negotiated session key. As a result, a key reinstallation attack as illustrated in Figure 4 would fail. Indeed, when the client no longer accepts plaintext handshake messages after installing the session key, the retransmitted (plaintext) message 3 in stage three of the attack is ignored.[1] As a result, no key reinstallation will take place. In [50], Vanhoef and Piessens proposed two techniques to overcome this problem, and both of them relied on device-specific race conditions.

Their first technique targeted Android devices. Against these devices, the attacker must wait with forwarding the first message 3 until it also received a retransmitted message 3. At that point, both message 3's are quickly send one after another to the victim. This triggers a race condition in the Wi-Fi chip of the victim, where the retransmitted plaintext message 3 is accepted for further processing, before the PTK is installed in response to the first message 3. Although the tested Android devices were vulnerable, other platforms such as macOS and OpenBSD were not affected by this race condition. Additionally, it may be hard to win (exploit) the race condition, and it is unclear how many Android devices are affected.

In the second technique, they trigger a key reinstallation during a rekey handshake [50]. Recall that a rekey handshake uses the same messages as in a normal handshake, except that all messages are now encrypted under the (old) session key. Similar to the previous technique, the adversary will wait with forwarding the first message 3 (now during a rekey), until it also received the retransmitted message 3. At that point, both message 3's are quickly sent one after another to the victim. This triggers a race condition where the retransmitted message 3 will be decrypted using the old session key, because the first message 3 has not yet been fully processed. Although this technique can be used to attack macOS and OpenBSD, it is rather impractical. The main problem is that few networks are configured to perform rekeys, and even if they are, a rekey happens only every few minutes to hours. For instance, Linux's hostapd AP disables rekeys of the PTK by default. Only the group key is periodically renewed. Additionally, the technique does not work against OpenBSD when the victim uses a Wi-Fi driver that employs software encryption. All combined, this technique to trigger key reinstallations in the 4-way handshake is of limited value in practice.

We conclude that, although key reinstallation attacks are feasible against most devices in theory, in practice the existing techniques can be tedious and impractical.

[1] This behaviour also causes a reliability issue: if message 4 is lost, the 4-way handshake will time out because the client drops the retransmitted plaintext message 3 [49]. Retransmitting message 3 is only useful when the original message 3 was lost.

3.2 Generating Encrypted Message 3's

We discovered a new technique to exploit key reinstallations against the 4-way handshake. In contrast to existing techniques, it does not rely on hard-to-win race conditions, but instead abuses power-save functionality in the AP. In particular, we manipulate the AP into sending retransmissions of message 3 that are encrypted under the newly negotiated session key. This encrypted message 3 will always be accepted by the client, even if it already installed the PTK. For example, unpatched versions of Android, macOS, and OpenBSD all accept the encrypted retransmitted message 3, and subsequently reinstall the session key. This demonstrates that an adversary no longer has to rely on hard-to-win race conditions to exploit vulnerable implementations of the 4-way handshake.

At a high level, we abuse power save functionality of 802.11 to make the AP temporarily buffer a retransmitted message 3. After this, we let the AP install the new PTK, and let it transmit all buffered frames. Since now a PTK is installed when the retransmitted message 3 is finally sent, it will be encrypted using it.

Figure 5 show the details of our attack. Again the adversary first establishes a multi-channel MitM position between the client and AP (see also Section 3.3). Note that the client is not explicitly drawn in Figure 5, its actions are clear from context. We assume the AP accepts older but unused handshake replay counters (e.g. hostapd). According to the standard all APs should do this, but in practice some APs only accept the latest replay counter [50]. In stage one of the attack, the adversary forwards the first three messages of the 4-way handshake without modification. However, message 4 is not forwarded to the AP. Instead, the adversary sends an empty (null) data frame to the AP with the sleep flag set (recall Section 2.1). As a result, the AP now believes the victim (client) is in sleep mode. This implies the AP will now buffer all frames sent towards the victim.

In stage three of the attack, the AP will retransmit message 3 since it has not yet received message 4 as a reply (see Figure 5). However, the 802.11 sublayer management entity (i.e. the kernel) thinks that the victim is in sleep mode. Therefore, it will queue the retransmitted message 3 instead of transmitting it. At this point, the adversary forwards the previously blocked message 4 to the client. Notice that the sleep bit of this message is set before forwarding it, which is possible because this information in the 802.11 header is not encrypted nor authenticated. When the AP receives message 4, the kernel will forward it to hostapd, and will keep thinking the client is asleep since the sleep bit is still set. Hostapd accepts this message 4, since its replay counter value $r + 1$ is higher than the one it previously received from the client. Note that other APs may deviate from this, and only accept replay counters that match the latest one used by the AP [50]. After validating message 4, hostapd will command the kernel into installing the newly negotiated PTK.

In stage 5 of the attack, the adversary will send an empty (null) data frame to the AP without the sleep flag set. This makes the AP mark the client as awake, causing the AP to transmit all buffered

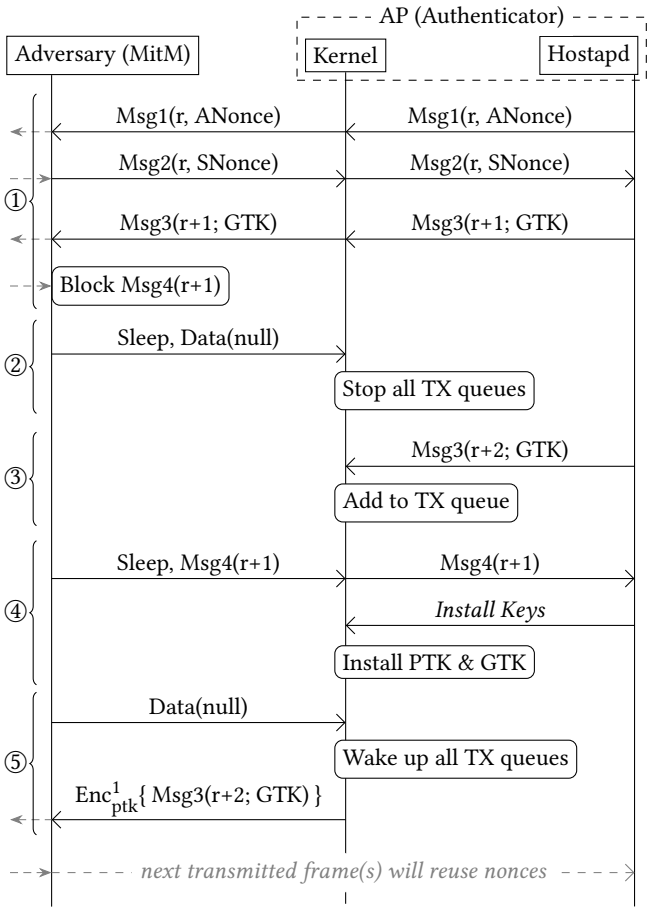

Figure 5: Making Linux's hostapd AP retransmit message 3of the 4-way handshake with link-layer encryption.

frames to the client. In particular, the retransmitted message 3 is now sent to the victim. And more importantly, it will be encrypted, because the kernel has already installed the new PTK. This encrypted message 3 can be forwarded to the victim, which will trigger a key reinstallation in all vulnerable implementations.

We tested this technique against hostapd version 2.6, which was running on Debian Jessie with Linux kernel 3.16. This confirmed the AP indeed encrypts the retransmitted message 3 under the new PTK. Since hostapd is used by both home routers and professional APs such as Aerohive, Cisco, Ubiquity, Aruba, and others, this attack is possible against many protected Wi-Fi networks [50].

Note that our new technique does not rely on tight timing constraints. Instead, it exploits logical features of power-save functionality in the AP. With our technique it is now also possible to attack OpenBSD even when it uses a Wi-Fi driver that employs software encryption. In contrast, the existing attacks based on race conditions were not able to accomplish this. Finally, we conjecture that more advanced power management functionality, such was WNM, can also be used to trick the AP into sending encrypted message 3's. However, we abused legacy power management functionality, because nearly all APs support this feature.

3.3 Improved MitM and Delaying Reinstalls

Performing most key reinstallation attacks, including the one against the 4-way handshake, requires a multi-channel MitM position [50]. Currently, obtaining this position requires special Wi-Fi equipment to jam all frames on a particular channel [47]. The jammer is used to prevent victims from connecting to the real AP. We propose a more practical method to obtain the MitM, which works based on Channel Switch Announcements (CSAs). In this method, the adversary forges CSAs to trick clients into switching to the desired (rouge) channel [27, 46]. This is more reliable then jamming certain channels, and does not require special Wi-Fi equipment. We successfully tested this approach against Android and Chromium.

A second limitation of existing attacks is that the key reinstallation is always triggered as soon as possible. In other words, once a retransmitted message 3 has been captured, it is immediately sent to the victim. However, it is also possible to delay the delivery of this message. This has as advantage that more frames will have been sent before the key reinstallation occurs, meaning more frames will also reuse nonces. This increases the impact of a key reinstallation. Delaying message 3 is even possible when it is encrypted. Indeed, handshake messages are typically sent using a fairly unique QoS priority. And since each priority channel has its own receive replay counter (recall Section 2.1), we can safely forward data frames to the victim with a higher replay counter, without affecting the reception of the encrypted message 3. We successfully tested delaying (an encrypted) message 3 against Linux, Android, iOS, and macOS.

4 ATTACKING THE FILS HANDSHAKE

In this section we study the Fast Initial Link Setup (FILS) handshake, and show that it is also vulnerable to key reinstallation attacks.

4.1 Background

The Fast Initial Link Setup (FILS) handshake was ratified in 2016 under the 802.11ai amendment [21]. Its purpose is to securely connect to an AP, while simultaneously initializing higher layer protocols by, for example, requesting and assigning an IP address.

With FILS, authentication can be performed using either a shared secret key, or a public key. The shared key can be a PMK, or a key called the re-authentication Root Key (rRK). A shared rRK can be establishing during the 802.1X authentication stage of a traditional handshake (recall Figure 2). Using an rRK instead of a PMK has the advantage that any AP on a particular network can obtain the rRK using the EAP Reauthentication Protocol (ERP) [12]. In contrast, a PMK can only be used with an AP that the client previously connect to, though a PMK does has the advantage that no ERP requests are necessary. We remark that for public key authentication, there are no guidelines on how or when to trust public keys.

An example execution of the FILS handshake is shown in stage one of Figure 6. The client initiates the handshake by sending an Authentication Request (AuthReq) to the AP. In our example, the frame contains an EAP-Initiate/Re-auth packet, indicating that the client wants to authenticate using a shared rRK. The AP extracts the EAP packet from the authentication request, and forwards it to a local (trusted) authentication server. In turn, the authentication server replies with an EAP-Finish/Re-auth packet as required by RFC 6696 [12]. When authentication was successful, this packet

contains the re-authentication Master Session Key (rMSK), which was derived from the rRK. The AP extracts the rMSK, and forwards the remaining content of the EAP packet to the client in an Authentication Response (AuthResp). This allows the client to derive the rMSK as well, meaning both the client and AP posses the rMSK.

Similar to the first two messages of the 4-way handshake, the authentication request and response frames transport a random SNonce and ANonce, respectively. If a rRK was used by the client, these nonces are combined with the rMSK to derive the PMK. In turn, the PMK is combined with both nonces and the MAC address of the client and AP to derive the PTK.

In the second part of the handshake, the client and AP exchange (re)association frames to confirm they negotiated the same PTK. More precisely, the client sends a (Re)Association Request (ReassoReq) with a Key Confirmation field containing a HMAC over the ANonce, SNonce, and the MAC address of both the client and AP. In turn, the AP replies using a (Re)Association Response (ReassoResp), which also contains a HMAC using the PTK over the same parameters (but ordered differently). All sensitive FILS fields in the (re)association response and request are encrypted using the PTK. Additionally, the (re)association response contains the (integrity) group key, which the client will install after receiving and processing the frame. Finally, once both the AP and client installed the PTK, encrypted data frames can be exchanged (see stage 2 of Figure 6).

Finally, clients can include a DHCP request in the (re)association request. The AP forwards this request to the local DHCP server, and returns the reply in the (re)association response.

4.2 Triggering a Key Reinstallation

The 802.11ai amendment does not define a state machine that describes how to implement the FILS handshake. More troublesome, it also does not specify when the PTK should be installed. Instead, it only specifies that the client should install the GTK after receiving (and decrypting) the (re)association response [21, §12.12.2.6.3]. This lack of rigorousness increases the chance of implementations having vulnerabilities. Nevertheless, we conjecture that most implementations will mimic the behaviour of the 4-way handshake, and install the PTK after completing the handshake. This means implementations will install the PTK after sending or receiving the FILS (re)association response (see Figure 6).

The standard also does not state what should happen when the AP receives a retransmitted FILS (re)association request. However, we can deduce the required reaction from context. In particular, we know that when a client has not received a (re)association response, it will retransmit the (re)association request. The goal of this is to make the AP send a new (re)association response. This means the AP must process and reply to retransmitted (re)association requests. Unfortunately, implementations may reinstall the PTK while doing so. That is, the AP may reinstall the PTK when receiving a retransmitted (re)association request during the FILS handshake.

An attacker can actively trigger these key reinstallations by replaying (re)association requests (see stage 3 of Figure 6). Unlike the attack against the 4-way handshake, a MitM position is not required. This is because none of the messages in the FILS handshake contain a replay counter, meaning we can trivially replay them. Moreover, plaintext (re)association frames must always be accepted, even if a

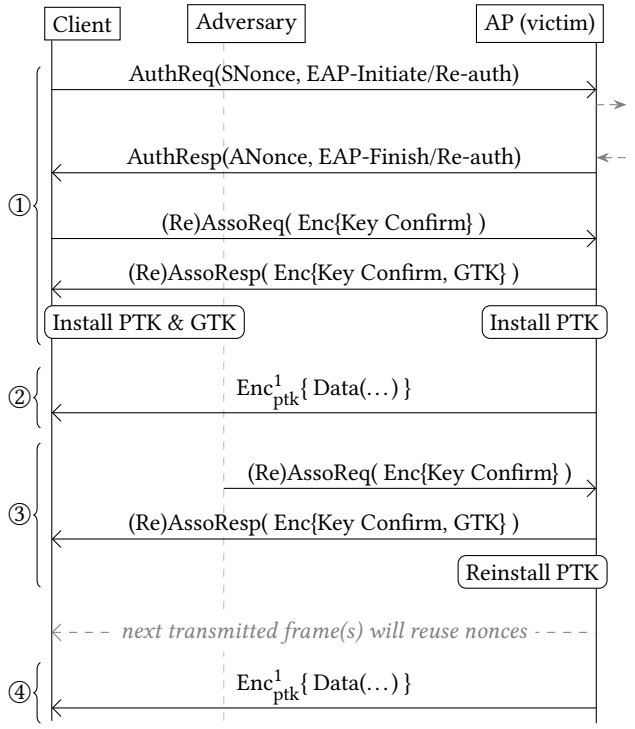

Figure 6: Key reinstallation attack against the Fast Initial Link Setup (FILS) handshake. A MitM position is not required, only the ability to sniff and replay frames.

session key is installed. After the AP replies with a retransmitted (re)association response, it will reinstall the PTK. Subsequent data frames sent by the AP now reuse nonces (see stage 4 in Figure 6), allowing an adversary to decrypt, replay, and possible forge frames.

4.3 Practical Evaluation

Because the FILS handshake was only ratified in December 2016, few APs currently support it. Nevertheless, we were able to test our attack against a development version of hostapd[2] on a Kali Linux distribution running kernel version 4.14. This confirmed that hostapd was vulnerable to the attack: merely replaying the (re)association requests triggered a key reinstallation. This shows vendors must be careful when implementing the FILS handshake, since otherwise trivial key reinstallation attacks are possible.

We reported the vulnerability, and it was fixed by not reinstalling keys when processing retransmitted FILS (re)association requests.

5 ATTACKING THE TPK HANDSHAKE

In this section we show how to perform key reinstallations against the Tunneled Direct-Link Setup (TDLS) PeerKey (TPK) handshake.

5.1 Background

The TDLS PeerKey (TPK) handshake is specified in amendment 802.11z, and provides a direct secure tunnel between two clients [24]. It is used by smart TVs, mobile phones, and tablets, to directly

[2]We tested our attack against commit 89c343e88 of 12 October 2017.

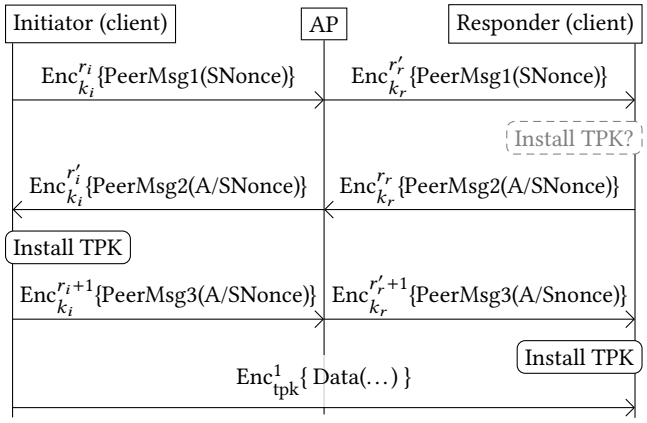

Figure 7: TPK handshake between two clients.

stream data between two devices. This avoids the overhead of having to pass all frames through the AP. Another use case of the TDLS tunnel is to transfer files to storage devices or printers. This tunnel forms an interesting target for attackers, because it may be used to exchange sensitive information (e.g. personal pictures and videos) without higher-layer protection.

Before executing the TPK handshake, there must already be a secure tunnel between each client and the AP. Once this tunnel is established, the TPK handshake can be performed as illustrated in Figure 7. Note that k_i and k_r represent the session keys of the secure tunnel between each client and AP. The client that initiates the handshake is called the initiator, and the other client is called the responder. The initiator starts by sending a TDLS Setup Request frame, which is represented by PeerMsg1, and contains a random SNonce. If the other client accepts the direct tunnel request, it generates a random ANonce, and combines it with the received SNonce to derive the TPK key. Although the standard specifies that the responder should install the TPK at this point, in practice implementations deviate from this, and install the TPK when receiving PeerMsg3. After generating the ANonce, the responder sends it to the initiator using a TDLS Setup Response frame (represented by PeerMsg2). Once the initiator learns the ANonce, it also derives and installs the TPK. Finally, the initiator sends a TDLS Setup Confirm frame, represented by PeerMsg3, to indicate that the handshake has successfully completed. If the responder did not yet install the TPK, it will install it after receiving PeerMsg3.

Note that the negotiated TPK only depends on the ANonce and SNonce, and not on a shared secret. This is not problematic, because we assume the AP is trustworthy and will not leak the nonces.

5.2 Key Reinstallation Attacks

Similar to the FILS handshake, the 802.11z amendment does not define a state machine describing the precise behaviour of the TPK handshake. In particular, it does not specify whether to retransmit TPK handshake messages when no reply was received. That said, the implementations we tested retransmit PeerMsg1 and PeerMsg2 if no corresponding PeerMsg2 or PeerMsg3, respectively, was received. Additionally, the tested implementations process retransmissions of PeerMsg1 and PeerMsg2.

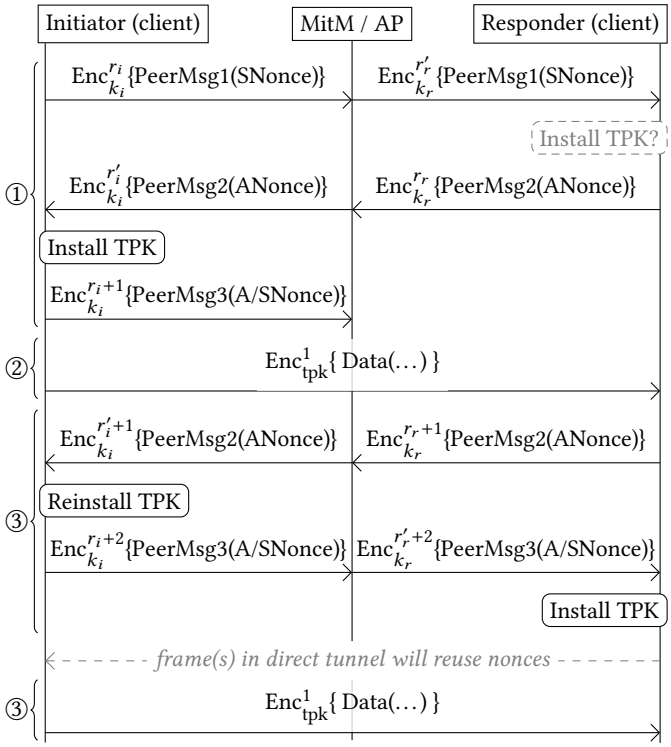

Figure 8: Key reinstallation attack against TPK. The MitM can be between the AP, and either the initiator or responder.

Given that implementations send and accept retransmitted TPK handshake messages, a key reinstallation attack can be mounted against them. In particular, Figure 7 illustrates an attack against the initiator of the TPK handshake. Similar to previous attacks, the adversary first establishes a multi-channel MitM between the initiator and responder. This can either be between the initiator and AP, or between the responder and AP. Once a MitM position is obtained, the adversary forwards the two first messages of the handshake without modification (see stage 1 of Figure 8). However, the third message is not forwarded to the responder. As a result, in stage two of the attack, the responder retransmits PeerMsg2. When the initiator receives this message, it will reinstall the TPK. This causes subsequent frames to reuse nonces, enabling the attacker to decrypt, replay, and possibly forge frames (see stage 3 in Figure 8).

If the responder installs the TPK after receiving PeerMsg1, and no longer accepts PeerMsg1's after receiving PeerMsg3, it is not vulnerable to key reinstallations. This is because the responder only starts sending data frames after receiving PeerMsg3, at which point key reinstallations are no longer possible. In the case the responder installs the PTK after receiving PeerMsg3, we conjecture that (most) implementations will not be vulnerable to key reinstallation attacks. This is because retransmissions of PeerMsg3 do not require a (new) reply, and therefore can simply be ignored by the responder.

5.3 Practical Evaluation

An overview of devices with a vulnerable TPK handshake is available under the vulnerability identifier CVE-2017-13086 [34]. One

interesting example is wpa_supplicant, because the version being used heavily influences the impact of the attack. Therefore we will discuss wpa_supplicant in more detail.

For versions 2.0 to 2.2 of wpa_supplicant, a key reinstallation attack against the initiator is possible precisely as illustrated in Figure 7. For versions 2.3 to 2.5, the initiator will reinstall the TPK, but immediately after the key reinstallation it tears down the direct link due to an internal bug. So although the TPK is being reinstalled, there is only a small time window when frames may reuse nonces, since the direct link is immediately teared down. Finally, in version 2.6 the initiator ignores retransmissions of PeerMsg2, meaning the initiator is not vulnerable to key reinstallations.

Surprisingly, we also found that all versions of wpa_supplicant accept retransmissions of PeerMsg3. These retransmissions cause the responder to reinstall the TPK. However, for version 2.3 and above, the key reinstallation is immediately followed by the teardown of the direct link due to an internal bug. Nevertheless, this means there is a small time window where the responder can be attacked. In practice this can be exploited by making the initiator retransmit PeerMsg3, which is possible by blocking the first PeerMsg3 from arriving at the responder. This causes the responder to retransmit PeerMsg2, which in turn makes the initiator send a new PeerMsg3. At this point the adversary can forward both PeerMsg3's to the responder, which triggers the key reinstallation. Note that this attack is only possible if the initiator correctly processes retransmitted PeerMsg2's (e.g. the attack is possible when the initiator uses version 2.0 to 2.2 of wpa_supplicant).

5.4 Handling Encrypted TPK Messages

To attack the TPK handshake, we must be able to distinguish encrypted TPK handshake messages from other encrypted frames. The first property that can be used for this is the length of TPK messages. In particular, PeerMsg1 has a data payload of 221 bytes, PeerMsg2 a payload of 107 bytes, and PeerMsg3 payload of 173 bytes. These numbers include the 8-byte LLC and SNAP header, but exclude the (predictable) overhead added by the encryption protocol. Additionally, normal data frames are sent between the client and the network gateway, but TPK messages are sent between two clients. As a result, this means we can also look at the source and destination MAC address of frames to recognize TPK messages.

Similar to the 4-way handshake, most implementations also use a fairly unique QoS priority to send TPK messages. This means retransmissions of PeerMsg2, which trigger the key reinstallation, can be arbitrary delayed. Indeed, even if the clients exchange data frames with higher replay counters, these will not influence the receive replay counter associated to the QoS channel of TPK messages. As a result, delayed TPK messages will still be accepted.

6 GROUP KEY REINSTALLATIONS AND WNM

In this section we use Wireless Network Management (WNM) features to both perform group key reinstallations, and to bypass 802.11's official countermeasure against key reinstallations. Additionally, we demonstrate implementation-specific vulnerabilities regarding the installation and usage of the group key.

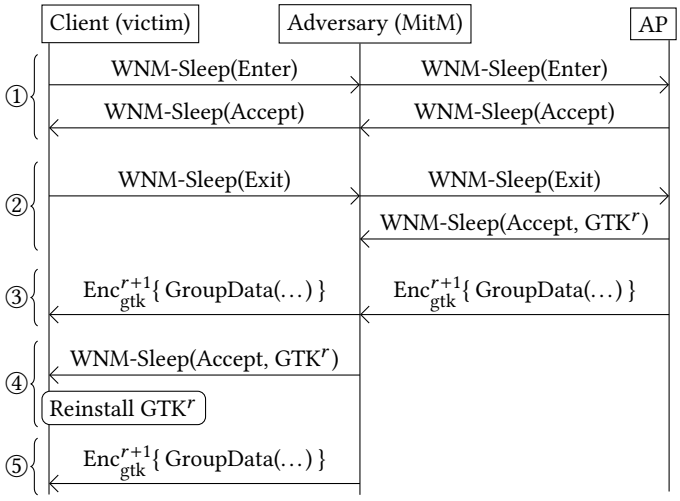

Figure 9: Abusing WNM-Sleep frames to reinstall the (integrity) group key, to then replay group-addressed frames.

6.1 Background

Wireless Network Management (WNM) features were ratified in the 802.11v amendment, and make it easier to manage clients [23]. Among other things, this includes the addition of WNM-Sleep mode. This is an extended power-save mode where clients can stay asleep much longer than previously possible. In particular, clients no longer need to wake up for group key updates.

A client enters WNM-Sleep mode by sending a WNM-Sleep Mode Request frame with an action field indicating the client wishes to enter sleep mode. The AP replies with a WNM-Sleep Mode Response frame that indicates whether the request was accepted or not. To exit sleep mode, the client sends a WNM-Sleep Mode Response frame indicating it is exiting sleep mode. Again the AP replies with a WNM-Sleep Mode Response frame, which now specifies whether the wakeup request was accept. More importantly, if Protected Management Frames (PMF) is enabled, this response will also contain the current (integrity) group key. Note that since 2018, PMF is required as part of WPA2 [55].

Based on security advisories of other WNM-Sleep mode vulnerabilities, and on our own disclosure process [35, 36], most mobile devices support WNM-Sleep functionality. For example, macOS [3], iOS [2], and Android [15], all support WNM-Sleep mode.

6.2 Group Key Reinstallations

We discovered that WNM-Sleep response frames can be abused to trigger key reinstallations. How this is accomplished is shown in Figure 9. In this attack, the adversary first establishes a multi-channel MitM position, and then waits until the client enters WNM-Sleep mode. We assume the client is using PMF, and that it keeps the current group key installed when entering sleep mode. This latter assumption is in line with 802.11v, which states that [23, §11.2.3.18.2]:

> "If RSN is used without management frame protection, the non-AP STA shall delete the GTKSA if the response indicates success."

In other words, the group key is only removed when PMF is not used. Since we do assume PMF is used, the group key is not removed. Interestingly, later versions of the 802.11 standard specify to always delete the (integrity) group key before going into WNM-Sleep [20]. Nevertheless, the implementations we tested do not remove the group key when going into WNM-Sleep mode.

At stage 2 of the attack, the adversary does not forward the WNM-Sleep response frame to the client (see Figure 9). Instead, the adversary waits until the AP sends group-addressed frames it wants to replay. Once such a frame is captured, it will forward it to the client, and will subsequently forward the (previously blocked) WNM-Sleep exit frame from the AP (see stage 4 of Figure 9). This causes the client to reinstall the group key. At this point the adversary can replay group-addressed frames (with a replay counter above r) that it previously captured (see stage 5 of the attack).

We tested the attack against wpa_supplicant version 2.6. For an overview of all other vulnerable devices, we refer to vulnerability identifiers CVE-2017-13087 and CVE-2017-13088 [35, 36].

6.3 Bypassing 802.11's Countermeasure

To prevent key reinstallations, the 802.11 standard was updated such that a key's associated parameters are not reset when it is being reinstalled (recall Section 2.5). Reinstallations are detected by comparing the new key with the currently installed one. However, this defense can be bypassed if an adversary can temporarily make the victim install a different key, before reinstalling the old one. Unfortunately, this is exactly what is possible by abusing interactions between EAPOL-Key and WNM-Sleep frames.

6.3.1 EAPOL-Key and WNM Frames. Figure 10 illustrates how combining WNM frames, with EAPOL-Key frames of the group key handshake, enables an attacker to bypass the official key reinstallation countermeasure of 802.11. The adversary first establishes a multi-channel MitM position, and then waits until the client enters and exits WNM-Sleep mode. Similar to the attack in Section 6.2, we safely assume the client will not delete the group key when going into sleep mode. In stage 2 of the attack, when the client exits sleep mode, the adversary does not forward the WNM-Sleep response to the client. We now wait until the AP performed two group key updates (stage 4 and 5 of the attack). Waiting for two updates is necessary because the first group key update uses KeyID 1, and only the second update overwrites the group key associated to KeyID 0. At this point the adversary can forward the WNM-Sleep response frame that was captured in stage 2 of the attack. Since the group key in the WNM-Sleep frame has KeyID 0, and differs from the newly installed key associated to KeyID 0, the client will (re)install the old key. In our case this (integrity) group key is installed with a replay counter of z, meaning we can now replay group-addressed frames with a higher replay counter. For example, in stage 7 the adversary replays the frame captured at stage 3 of the attack.

Our attack requires us to wait until the AP performed two group key updates. In case the AP is using strict group rekeying, an adversary can trigger these group key updates by forcibly disconnecting another client. Note that disconnecting a client when PMF is enabled is still possible by forging channel switch announcements [27].

Figure 13 in the Appendix contains another variant of the attack. There, WNM frames are used to temporarily install a new (integrity) group key, while group handshake messages are abused to reinstall the old key. Forwarding the group key handshake message in stage 6 of the attack is possible even if data frames with a higher replay counter have previously been sent to the client. This is because the group key messages use a fairly unique QoS priority. Yet another variant is shown in Figure 14 of the Appendix. Here EAPOL-Key frames of the 4-way handshake are combined with WNM frames to bypass 802.11's countermeasure and reinstall the group key.

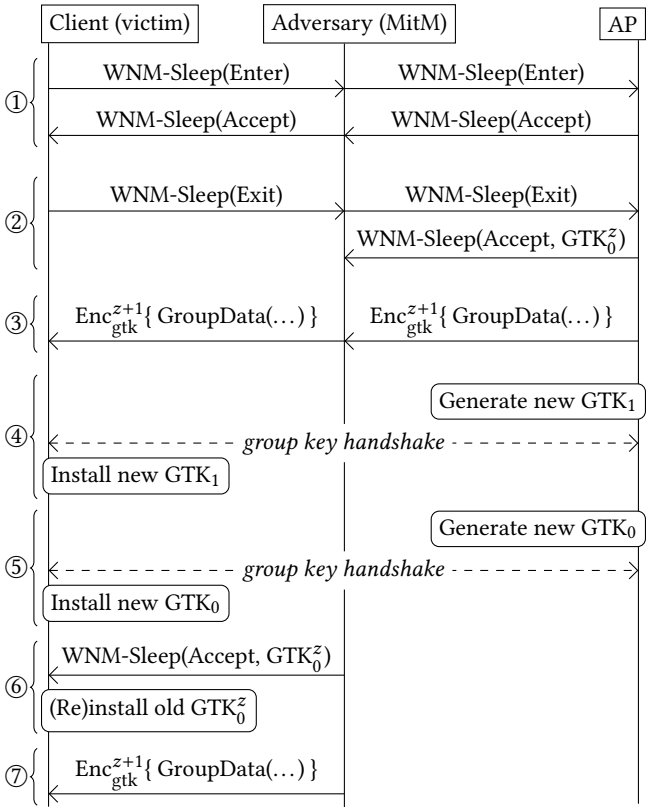

Figure 10: Interleaving EAPOL-Key and WNM-Sleep frames to bypass key reinstallation countermeasures. More precisely, we first (re)install an old (integrity) group key, after temporarily installing a different key. In this case, the reinstallation itself is caused by a WNM-Sleep frame.

6.3.2 Improved Countermeasure. A naive solution is to track all group keys that have previously been installed. This is impractical because a network can update the group key an arbitrary number of times, meaning a large number of keys would need to be tracked. Moreover, against APs that use strict group rekeying, a new group key is generated whenever a client leaves the network.

A more efficient defense is to track the latest (integrity) group key installed in response to an EAPOL-Key frame, and the latest (integrity) group key installed in response a WNM-Sleep frame. This means that two normal group keys are saved, and two integrity group keys are also saved. When now receiving a new key in either an EAPOL-Key or WNM-Sleep frame, the new (integrity) group key must only be installed if it differs from both of the two

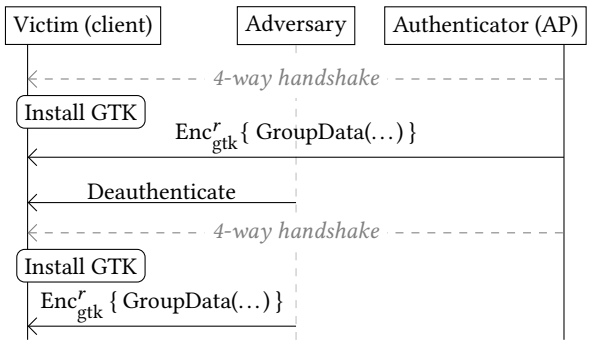

Figure 11: Replaying group-addressed frames against clients that always install the GTK with an all-zero replay counter.

saved (integrity) group keys. Additionally, we require that the client disconnects from the network if it did not receive a WNM-Sleep response frame when exiting sleep mode. We also recommend that a client deletes the current (integrity) group key before entering WNM-Sleep mode. All combined, this prevents the attacks in Figures 10, 13, and 14. Intuitively, the idea behind this defense relies on the replay counters used in WNM and group key frames. That is, an adversary can only use the last not-yet-received WNM or EAPOL-Key frame during an attack. Since we track the last key installed in response to either frame, and we cannot let these frames interact in different manners using older frames (due to out-of-date replay counters), attacks are prevented. A formal analysis of our new countermeasure is unfortunately out of scope for this paper.

We informed the 802.11 working group of our new defense, but they have not incorporated it into the standard. In contrast, Hostapd has implemented this defense in its development version.

6.4 Improper GTK Installation and Usage

During our analysis of group key installations, we also discovered two common types of implementation-specific vulnerabilities. The first one is rather trivial. Namely, certain devices always accept replayed group-addressed frames. In total we tested 18 different devices, and found that half of them accept replayed group-addressed frames (see Table 1). We conjecture this is usually caused by a faulty hardware decryption engine in the Wi-Fi chip.

A related flaw is that some devices always install the group key with an all-zero replay counter (see Table 1). Recall that normally the group key must be installed with a given replay counter. However, these devices ignore this value. An adversary can abuse this by forcibly disconnecting the client, after which it will reconnect to the network, and (re)install the group key using an all-zero replay counter. This allows the adversary to replay group-addressed frames to the victim (see Figure 11). Interestingly, the integrity group key was always installed with the correct replay counter (see Table 1).

7 IMPLEMENTATION-SPECIFIC ANALYSIS

This section discusses bugs that we discovered while analyzing key reinstallation patches, and inspecting implementations in general. For example, we found implementations that reused the SNonce or ANonce in the 4-way handshake, routers that accept replayed handshake messages, and other implementation-specific flaws.

Table 1: Implementation-specific flaws. The 2nd column indicates if devices accept replayed group-addressed frames. The 3rd whether it always installs the group key with an all-zero replay counter. The last column if the integrity group key is always installed with an all-zero replay counter. For laptop and USB devices under the gray line, L and W denote the attack only works on Linux or Windows, respectively.

Implementation	Replay	Flawed GTK	Flawed IGTK
Galaxy S3 LTE	No	Yes	No
Nexus 5X	Yes	? [a]	No
iOS 11.2.6 (iPad)	No	Yes	No
macOS 10.13.4	No	Yes	No
Linksys RE7000	Yes	? [a]	No
Ralink 802.11n USB	No	No	No
Sitecom NIC	Yes	? [a]	No
TL-WN722N	No	No	No
RTL8188CUS	W	? [a]	No
DWL-AG132	W	? [a]	No
AWUS036H	W	? [a]	No
AWUS036NH v.2	Yes	? [a]	No
Intel 7260	No	No	No
AWUS036NHA	No	No	No
WNDA3200	No	No	No
Belkin F5D8053 v3	Yes	? [a]	No
TWFM-B003D	No	No	No
ZyXel NWD6505	W	? [a]	No

[a] We are unable to test this attack because the device always accepts replayed group-addressed frames (see column two).

7.1 ANonce and SNonce Reuse

When analyzing the key reinstallation patch of macOS High Sierra 10.13.2 beta, we confirmed that it prevented key reinstallation attacks against the 4-way handshake as shown in Figure 4 (even when using encrypted message 3's in the attack). However, subsequent tests showed that during a rekey of the session key (recall Section 2.3), macOS reused the previous SNonce value. Interestingly, in a related discovery, Malinen found that hostapd APs reused the ANonce during rekeys of the session key [31]. This means that if a macOS device was connected to an AP running hostapd, and the session key was periodically refreshed, they would constantly negotiate the same key. As a result, this key would constantly be reinstalled, leading to nonce and keystream reuse. This makes it possible to decrypt old (passively captured) traffic.

It is also possible to attack a client that reuses the SNonce, even when the real AP does not reuse the ANonce. Figure 12 shows how this can be accomplished. The adversary starts by obtaining a

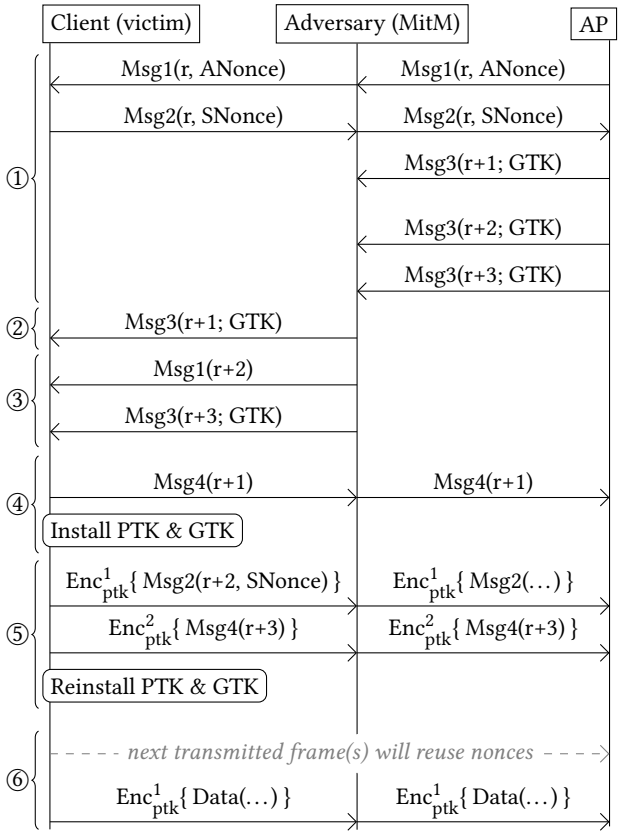

Figure 12: Replaying an old handshake against a client that reuses the previous SNonce during a rekey. This attack variant assumes the client accepts plaintext handshake messages that are sent immediately after the first message 3.

multi-channel MitM position, and then blocks the first three message 3's from arriving at the client. In the second stage of the attack, the adversary forwards the first message 3 to complete the initial 4-way handshake with the client. Additionally, the adversary sends a forged message 1, and sends the previously blocked message 3 with replay counter $r + 3$ to the client (see stage 3 in Figure 12). These two latter messages will form the second 4-way handshake. In stage 4, the client receives the first message 3, and will complete the handshake and install the session key. Assuming the client still accepts plaintext handshake messages when they are sent immediately after the first message 3, it will now process the messages sent during stage 3 of the attack. As a result, it will reply using a new message 2 and 4, completing the second 4-way handshake. Remark that the client accepts this new handshake because it is still using the same SNonce value, and the messages sent by the adversary all use the old ANonce value. At this point the client will reinstall the old session key, leading to nonce reuse (see stage 6 in Figure 12).

If the client never accepts plaintext handshake messages after installing the PTK, we can use the technique presented in Section 7.2 to generate arbitrary encrypted handshake messages. However, in contrast to the attack of Figure 12, and to the technique of Section 3.2 to generate encrypted message 3's, the technique presented in Section 7.2 works against fewer network configurations.

We reported the SNonce reuse vulnerability to Apple, and it has been fixed in macOS High Seirra 10.13.3 and above.

Table 2: Routers tested for a replay of message 4. Column two contains the vendor of the Wi-Fi chip(s) in the router.

Router	Wi-Fi Driver	Vulnerable
Linksys WAG320N	Broadcom	No
Galaxy S3 i9305	Broadcom	No
Cisco Aironet 1130 AG	Unknown	No
Linksys EA7500	MediaTek	No
RE7000	MediaTek	No
RT-AC51U	MediaTek	Yes
TP-Link RE370K	MediaTek	Yes

7.2 Generating Encrypted Message 1's

In an addendum to their paper, Vanhoef and Piessens mention that version 2.6 of wpa_supplicant can be tricked into installing an all-zero key during the 4-way handshake.[3] To accomplish this, an adversary must inject a forged message 1, with the same ANonce as used in the original message 1, before forwarding a retransmitted message 3 to the victim. If we want to perform the attack against Android without relying on hard-to-win race conditions, we need to generate both encrypted message 1's and message 3's. We showed in Section 3.2 how to generate encrypted message 3's, however, this technique cannot be used to generate encrypted message 1's.

An adversary can generate encrypted message 1's when the victim is connected to an enterprise network, and the adversary can also connect to this network. In that situation, the adversary lets the client connect to the AP, after which it also connects to the AP. Then it sends the forged message 1 through the AP to the victim. The AP will encrypt the frame when sending it to the victim. Moreover, the adversary can abuse A-MSDU frames against Linux 4.8 and below to spoof the source address of message 1, such that it appears to be sent by the AP (recall Section 2.1). Admittedly this attack is not applicable in all scenarios, but it does show defenders must take into account that an adversary can generate both encrypted message 1's and message 3's.

7.3 Replaying a Handshake's Last Message

We also investigated whether some devices accept retransmissions of the last message in a handshake. Note that, according to the standard, these retransmissions should be ignored, as the handshake already completed. Nevertheless, some may accept them due to implementation bugs. We first tested whether replaying message 4 of the 4-way handshake triggers a key reinstallation in the AP. This was done by replaying the message using the same replay counter as message 3, and by using an increased one. Additionally, we performed our tests with both plaintext and encrypted message 4's.

We tested all devices in Table 2, and discovered that the default firmware of the RT-AC51U and TP-Link RE370K accepts replayed

[3] See https://www.krackattacks.com/

message 4's. Upon further inspection, we found that the MediaTek driver for both the 2.4 and 5 GHz Wi-Fi chips in these devices has an invalid state check in the function that processes message 4's (see Listing 1). The function starts by checking if it has already received message 2 (see line 6). If not, it discards the message. However, this condition does not check whether a message 4 has already been received (i.e. whether WpaState equals AS_PTKINITDONE). As a result, replays of message 4 are accepted. Moreover, an adversary can simply replay the original plaintext message 4, meaning a MitM is not required. Rather worrisome, these Wi-Fi chips are present in more than 100 different devices, ranging from APs, wireless cameras, wireless network extenders, home automation switches, NAS devices, smart power plugs, and so on [58]. Unless these devices use a different driver from the ones we tested, they are all vulnerable.

We also tested if the last message in the FT handshake can be replayed towards the client. This did not uncover any vulnerabilities in wpa_supplicant, Windows, or iOS. Other clients did not yet support FT. Once more devices support the FT handshake, and support other new handshakes such as FILS, it will be interesting to perform a more extensive analysis of implementations.

We notified MediaTek of the vulnerability. They stated it will be fixed in a new release of their driver. However, at the time of writing, a patch for our RT-AC51U router was not yet available.

7.4 Forging Message 3

Another interesting use case revolves around a recently discovered vulnerability in wpa_supplicant [51]. In particular, wpa_supplicant incorrectly handled EAPOL-Key frames that have the Encrypted flag set, but not the MIC flag. Against version 2.6 and lower of wpa_supplicant, this malformed EAPOL-Key frame was treated as message 1 of the 4-way handshake, and could be abused to mount a decryption oracle attack to e.g. decrypt the group key [51]. However, in the development version of wpa_supplicant, the malformed frame is treated as message 3 of the 4-way handshake. This change in behaviour was caused by a commit that added support for AEAD ciphers in the 4-way handshake [29]. We can abuse this new behaviour to forge message 3's, and trigger key reinstallations against the 4-way handshake without needing a MitM, as follows:

(1) Capture a valid message 3 sent by the AP.
(2) Unset the MIC flag in the EAPOL-Key header, and increase the replay counter (recall Section 2.3).
(3) Inject the modified message 3 towards the client.

The development version of wpa_supplicant will not verify the authenticity of the EAPOL-Key frame, but will still process it as message 3 of the 4-way handshake. And since we increased the replay counter, the message is treated as a new one. Hence, the client will reinstall the session key after replying with a new message 4. Additionally, wpa_supplicant will decrypt the key data field and reinstall the (integrity) group key. Moreover, because the full EAPOL-Key frame is not authenticated, we can set the replay counter of the transported group key to any value we desire (since the replay counter of the group key is saved in the RSC field, which in our scenario is not authenticated). This allows an adversary to reset the replay counter to zero, allowing replays of group-addressed frames, or it can be set to a very high value, leading to a denial-of-service attack where the victim will drop all group-addressed traffic.

Listing 1: Simplified code of the MediaTek driver in the RT-AC51U that is vulnerable to a replay of message 4.

```
1  enum _ApWpaState { /* ... */
2    AS_PTKSTART, AS_PTKINIT_NEGOTIATING, AS_PTKINITDONE }
3
4  VOID PeerPairMsg4Action(PRTMP_ADAPTER pAd, MAC_TABLE_ENTRY *pEntry,
5    MLME_QUEUE_ELEM *pMsg) {
6      if (pEntry->WpaState < AS_PTKINIT_NEGOTIATING)
7          return;
8      if (!PeerWpaMessageSanity(pMsg, EAPOL_PAIR_MSG_4, pEntry))
9          return;
10
11     WPAInstallPairwiseKey(pAd, pEntry->func_tb_idx, pEntry, TRUE);
12     pEntry->WpaState = AS_PTKINITDONE;
13 }
```

To carry out the above attack in practice, the victim needs to accept plaintext EAPOL-Key messages after installing the session key. Although this is the case for several Linux platforms, this does not hold for e.g. Android [50]. Nevertheless, the technique of Section 7.2 can be used to attack a client even if it only accepts encrypted EAPOL-Key frames after installing the session key.

7.5 Discussion: Network Protocol Gadgets

One common theme in this work is that seemingly innocent bugs, and even protocol features, can act as essential gadgets when trying to exploit (implementation-specific) protocol bugs. For example, on its own the technique of Section 3.2 to generate encrypted handshake messages is innocent. However, this technique allowed us to perform key reinstallation attacks against OpenBSD drivers that use software encryption. Similarly, the A-MSDU bug in Linux that allows an adversary to spoof the source and destination address, has little impact on its own. Unfortunately, in Section 7.2 and 7.4, this bug proved essential to trigger key reinstallations in certain versions of wpa_supplicant. As another example, having separate receive replay counters for each QoS channel is also not a security risk on its own. However, this protocol feature allowed us to attack the group key and TPK handshake. In other words, seemingly innocent implementation bugs or protocol features can act as gadgets that enable exploitation of other vulnerabilities.

The most intriguing example is that users of wpa_supplicant noticed that some versions installed an all-zero key when processing a retransmitted message 3 [30]. However, they did not realize an adversary could actively trigger these retransmissions, and only treated this as an innocent bug. In a sense they almost discovered key reinstallation attacks, but failed to realize the capabilities of an adversary, and therefore did not realize this was exploitable.

These observations teach us that it can be very difficult to determine whether a protocol bug is exploitable or not. In other words, one needs an expert understanding of all available gadgets, i.e., related protocol features and implementations bugs. Only when having this knowledge is it possible to accurately determine whether, and under which conditions, a protocol bug is exploitable.

8 RELATED WORK

Vanhoef and Piessens introduced the key reinstallation attack, and demonstrated that it affected the 4-way, group key, PeerKey, and

FT handshake [50]. Vanhoef et al. also proposed a defense against the multi-channel MitM position, which is required in several of theirs and ours attacks [46]. There are no other known attacks where an adversary can actively trigger nonce reuse in a similar manner. Although a power failure can also lead to nonce reuse [60], and some TLS servers have been found to use static nonces [7], an adversary is not able to actively trigger nonce reuse in these cases. Additionally, some network cards always initialize the WEP IV to zero, making reuse of small IVs more likely [8, 9]. However, an adversary cannot actively trigger these IV initializations.

He and Mitchell formally analyzed the 4-way handshake, and discovered a denial-of-service vulnerability [18, 32]. This resulted in the standardization of a slightly improved 4-way handshake [20]. He et al. continued to analyze the 4-way handshake, and proved its correctness in 2005 [19]. However, implementations of the 4-way handshake were nevertheless vulnerable to downgrade attacks [48].

Researchers successfully attacked the older (WPA-)TKIP encryption protocol. The first type of works exploit its weak message integrity code, allowing an adversary to decrypt and forge packets sent towards the client [16, 33, 43, 45, 47]. The second type of attacks exploit the RC4 encryption algorithm used in TKIP [1, 37, 38]. These attacks against RC4 enable an adversary to decrypt secrets that are repeatedly transmitter over the network. Nowadays, TKIP is deprecated by the Wi-Fi Alliance due to its security issues [54].

Other attacks target the old Wired Equivalent Privacy (WEP) protocol. Here Fluhrer et al. exploited weaknesses in the key schedule of RC4 to recover the secret key [14]. This attack was later demonstrated in practice [42], and over time improved by several researchers [40, 44]. Other attacks against WEP were also discovered, for example, fragmentation of frames was used to break WEP [6].

Researchers also discovered flaws in implementations and technology surrounding Wi-Fi. For example, they found flaws in Wi-Fi Protected Setup (WPS) [53], vulnerable drivers [4, 11], predictable random number generators [48], predictable pre-shared keys [28], insecure enterprise authentication [10, 13, 39, 59], implementation bugs in handshakes [41, 49, 52], and so on.

9 CONCLUSION

Our results show that preventing key reinstallation attacks is harder than initially assumed. For example, we were able to bypass the official countermeasure of the 802.11 standard and reinstall the group key, by combining WNM-Sleep frames with EAPOL-Key frames. Additionally, we showed that the FILS and TPK handshakes are vulnerable to key reinstallation attacks. Apart from attacks against the standard, we also discovered novel implementation-specific vulnerabilities that facilitate key (re)installation attacks.

We believe the main reason vulnerabilities are still present is because the 802.11 standard is large, is continually being extended with new features, and requires domain-specific knowledge to understand. These obstacles can be avoided by having up-to-date and high-level descriptions (or formal models) of all security-related features of 802.11. This would make it easier to reason about its design, and test the correctness of implementations. Another option would be to simplify the standard, which would again make it easier to analyze. Additionally, we believe the Wi-Fi Alliance should not only test products for interoperability during their certification process, but also fuzz them for vulnerabilities.

ACKNOWLEDGMENTS

This research is partially funded by the Research Fund KU Leuven. Mathy Vanhoef holds a Postdoctoral fellowship from the Research Foundation Flanders (FWO).

REFERENCES

[1] Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. 2013. On the Security of RC4 in TLS and WPA. In *USENIX Security*.
[2] Apple. 2017. About the security content of iOS 10.3.3. Retrieved 18 August 2018 from https://support.apple.com/en-us/HT207923.
[3] Apple. 2017. About the security content of macOS Sierra 10.12.6, Security Update 2017-003 El Capitan, and Security Update 2017-003 Yosemite. Retrieved 18 August 2018 from https://support.apple.com/en-us/HT207922.
[4] Gal Beniamini. 2017. Over The Air: Exploiting Broadcom's Wi-Fi Stack. Last retrieved 7 May from https://googleprojectzero.blogspot.be/2017/04/over-air-exploiting-broadcoms-wi-fi_4.html.
[5] Johannes Berg. 2016. cfg80211: add ability to check DA/SA in A-MSDU decapsulation. Linux commit `8b935ee2e`.
[6] Andrea Bittau, Mark Handley, and Joshua Lackey. 2006. The Final Nail in WEP's Coffin. In *IEEE S&P*.
[7] Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. 2016. Nonce-Disrespecting Adversaries: Practical Forgery Attacks on GCM in TLS. In *USENIX WOOT*.
[8] Nikita Borisov, Ian Goldberg, and David Wagner. 2001. Analysis of 802.11 Security, or Wired Equivalent Privacy Isn't. In *Mac Crypto Workshop*.
[9] Nikita Borisov, Ian Goldberg, and David Wagner. 2001. Intercepting mobile communications: the insecurity of 802.11. In *MobiCom*.
[10] Sebastian Brenza, Andre Pawlowski, and Christina Pöpper. 2015. A practical investigation of identity theft vulnerabilities in eduroam. In *WiSec*.
[11] Laurent Butti and Julien Tinnes. 2008. Discovering and exploiting 802.11 wireless driver vulnerabilities. *Journal in Computer Virology* 4, 1 (2008), 25–37.
[12] Z. Cao, B. He, Y. Shi, Q. Wu, and G. Zorn. 2012. *EAP Extensions for the EAP Re-authentication Protocol (ERP)*.
[13] Aldo Cassola, William Robertson, Engin Kirda, and Guevara Noubir. 2013. A Practical, Targeted, and Stealthy Attack Against WPA Enterprise Authentication. In *NDSS Symp*.
[14] Scott Fluhrer, Itsik Mantin, and Adi Shamir. 2001. Weaknesses in the key scheduling algorithm of RC4. In *SAC*.
[15] Google. 2017. Android Security Bulletin–September 2017. Retrieved 18 August 2018 from https://source.android.com/security/bulletin/2017-09-01.
[16] Finn M. Halvorsen, Olav Haugen, Martin Eian, and Stig F. Mjølsnes. 2009. An Improved Attack on TKIP. In *14th Nordic Conference on Secure IT Systems (NordSec '09)*. 13.
[17] Dan Harkins and Jouni Malinen. 2017. Addressing the Issue of Nonce Reuse in 802.11 Implementations. Retrieved 8 February 2018 from https://mentor.ieee.org/802.11/dcn/17/11-17-1602-03-000m-nonce-reuse-prevention.docx.
[18] Changhua He and John C Mitchell. 2004. Analysis of the 802.1 i 4-Way Handshake. In *WiSe*. ACM.
[19] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C Mitchell. 2005. A modular correctness proof of IEEE 802.11i and TLS. In *CCS*.
[20] IEEE Std 802.11. 2016. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Spec*.
[21] IEEE Std 802.11ai. 2016. *Amendment 1: Fast Initial Link Setup*.
[22] IEEE Std 802.11i. 2004. *Amendment 6: Medium Access Control (MAC) Security Enhancements*.
[23] IEEE Std 802.11v. 2011. *Amendment 8: IEEE 802.11 Wireless Network Management*.
[24] IEEE Std 802.11z. 2008. *Amendment 7: Extensions to Direct-Link Setup (DLS)*.
[25] Jakob Jonsson. 2002. On the security of CTR+ CBC-MAC. In *SAC*.
[26] Antoine Joux. 2006. Authentication failures in NIST version of GCM. *Retrieved 8 May 2017 from http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/Joux_comments.pdf* (2006).
[27] Bastian Könings, Florian Schaub, Frank Kargl, and Stefan Dietzel. 2009. Channel switch and quiet attack: New DoS attacks exploiting the 802.11 standard. In *LCN*.
[28] Eduardo Novella Lorente, Carlo Meijer, and Roel Verdult. 2015. Scrutinizing WPA2 password generating algorithms in wireless routers. In *USENIX WOOT*.
[29] Jouni Malinen. 2015. FILS: Use AEAD cipher to check received EAPOL-Key frames (STA). Hostap commit `0ab1dd010`.
[30] Jouni Malinen. 2015. Fix TK configuration to the driver in EAPOL-Key 3/4 retry case. Hostap commit `ad00d64e7`.

[31] Jouni Malinen. 2017. Fix PTK rekeying to generate a new ANonce. Retrieved 7 May from https://w1.fi/security/2017-1/0005-Fix-PTK-rekeying-to-generate-a-new-ANonce.patch.

[32] John Mitchell and Changhua He. 2005. Security Analysis and Improvements for IEEE 802.11i. In *NDSS*.

[33] Masakatu Morii and Yosuke Todo. 2011. Cryptanalysis for RC4 and Breaking WEP/WPA-TKIP. *IEICE Transactions* (2011), 2087–2094.

[34] National Vulnerability Database. 2017. CVE-2017-13086 Detail. Retrieved 6 May 2018 from https://nvd.nist.gov/vuln/detail/CVE-2017-13086.

[35] National Vulnerability Database. 2017. CVE-2017-13088 Detail. Retrieved 6 May 2018 from https://nvd.nist.gov/vuln/detail/CVE-2017-13087.

[36] National Vulnerability Database. 2017. CVE-2017-13088 Detail. Retrieved 6 May 2018 from https://nvd.nist.gov/vuln/detail/CVE-2017-13088.

[37] Kenneth G Paterson, Bertram Poettering, and Jacob CN Schuldt. 2014. Big Bias Hunting in Amazonia: Large-Scale Computation and Exploitation of RC4 Biases. In *AsiaCrypt*.

[38] Kenneth G. Paterson, Jacob C. N. Schuldt, and Bertram Poettering. 2014. Plaintext Recovery Attacks Against WPA/TKIP. In *FSE*.

[39] Pieter Robyns, Bram Bonné, Peter Quax, and Wim Lamotte. 2014. Short paper: exploiting WPA2-enterprise vendor implementation weaknesses through challenge response oracles. In *WiSec*.

[40] Pouyan Sepehrdad, Petr Susil, Serge Vaudenay, and Martin Vuagnoux. 2015. Tornado Attack on RC4 with Applications to WEP & WPA. Cryptology ePrint Archive, Report 2015/254.

[41] Christopher McMahon Stone, Tom Chothia, and Joeri de Ruiter. 2018. Extending Automated Protocol State Learning for the 802.11 4-Way Handshake. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*.

[42] Adam Stubblefield, John Ioannidis, Aviel D Rubin, et al. 2002. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. In *NDSS*.

[43] Erik Tews and Martin Beck. 2009. Practical attacks against WEP and WPA. In *WiSec*.

[44] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. 2007. Breaking 104 bit WEP in less than 60 seconds. In *JISA*.

[45] Yosuke Todo, Yuki Ozawa, Toshihiro Ohigashi, and Masakatu Morii. 2012. Falsification Attacks against WPA-TKIP in a Realistic Environment. *IEICE Transactions* 95-D, 2 (2012).

[46] Mathy Vanhoef, Nehru Bhandaru, Thomas Derham, Ido Ouzieli, and Frank Piessens. 2018. Operating Channel Validation: Preventing Multi-Channel Man-in-the-Middle Attacks Against Protected Wi-Fi Networks. In *WiSec*.

[47] Mathy Vanhoef and Frank Piessens. 2014. Advanced Wi-Fi attacks using commodity hardware. In *ACSAC*.

[48] Mathy Vanhoef and Frank Piessens. 2016. Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys. In *USENIX Security*.

[49] Mathy Vanhoef and Frank Piessens. 2017. Denial-of-Service Attacks Against the 4-way Wi-Fi Handshake. In *9th International Conference on Network and Communications Security (NCS)*.

[50] Mathy Vanhoef and Frank Piessens. 2017. Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2. In *CCS*.

[51] Mathy Vanhoef and Frank Piessens. 2018. Symbolic Execution of Security Protocol Implementations: Handling Cryptographic Primitives. In *USENIX WOOT*.

[52] Mathy Vanhoef, Domien Schepers, and Frank Piessens. 2017. Discovering logical vulnerabilities in the Wi-Fi handshake using model-based testing. In *ASIA CCS*. ACM.

[53] Stefan Viehböck. 2011. Brute forcing Wi-Fi protected setup. Retrieved February 1 2017 from http://packetstorm.foofus.com/papers/wireless/viehboeck_wps.pdf.

[54] Wi-Fi Alliance. 2015. *Technical Note: Removal of TKIP from Wi-Fi Devices*.

[55] Wi-Fi Alliance. 2018. Discover Wi-Fi: Security. Retrieved 8 May 2018 from https://www.wi-fi.org/discover-wi-fi/security

[56] Wi-Fi Alliance. 2018. Product Finder Results: TLDS Certified Products. Retrieved 16 August 2018 from https://www.wi-fi.org/product-finder-results?certifications=38

[57] Wi-Fi Alliance. 2018. WPA3 Specification Version 1.0. Retrieved 18 August 2017 from https://www.wi-fi.org/file/wpa3-specification-v10.

[58] WikiDevi. 2018. MediaTek MT7620. Retrieved 27 March 2018 form https://wikidevi.com/wiki/MediaTek_MT7620.

[59] Joshua Wright. 2003. Weaknesses in LEAP challenge/response. In *DEF CON*.

[60] Erik Zenner. 2009. Nonce Generators and the Nonce Reset Problem. In *International Conference on Information Security*.

APPENDIX

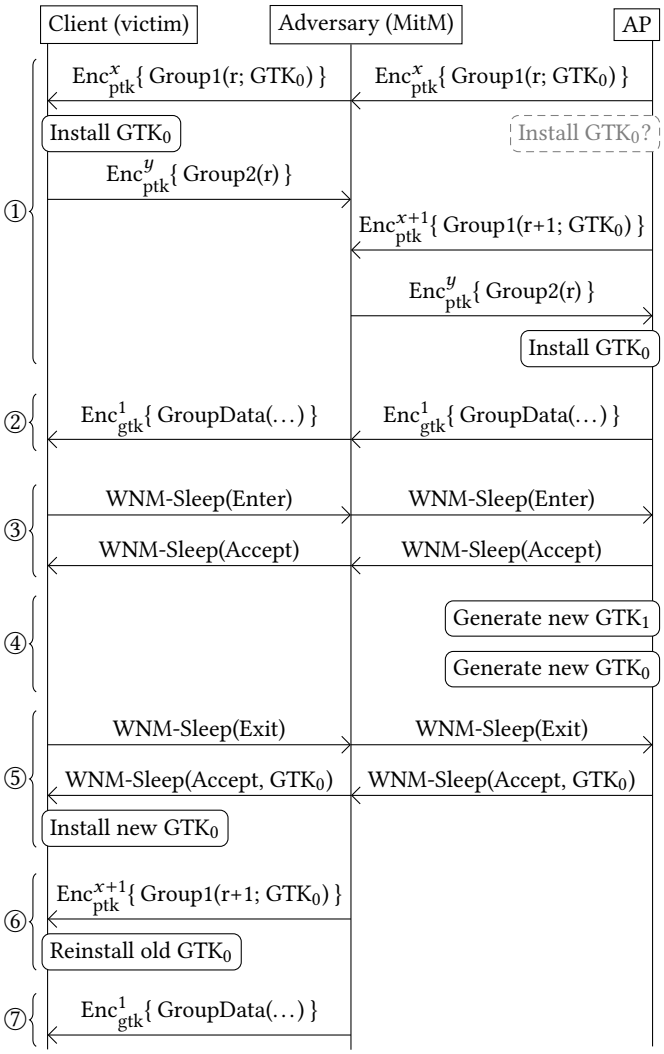

Figure 13: Abusing the interaction between the group key handshake and WNM-Sleep frames to bypass key reinstallation countermeasures. More precisely, we first (re)install an old (integrity) group key, after temporarily installing a different key. In this case, the reinstallation itself is caused by an EAPOL-Key frame of the group key handshake.

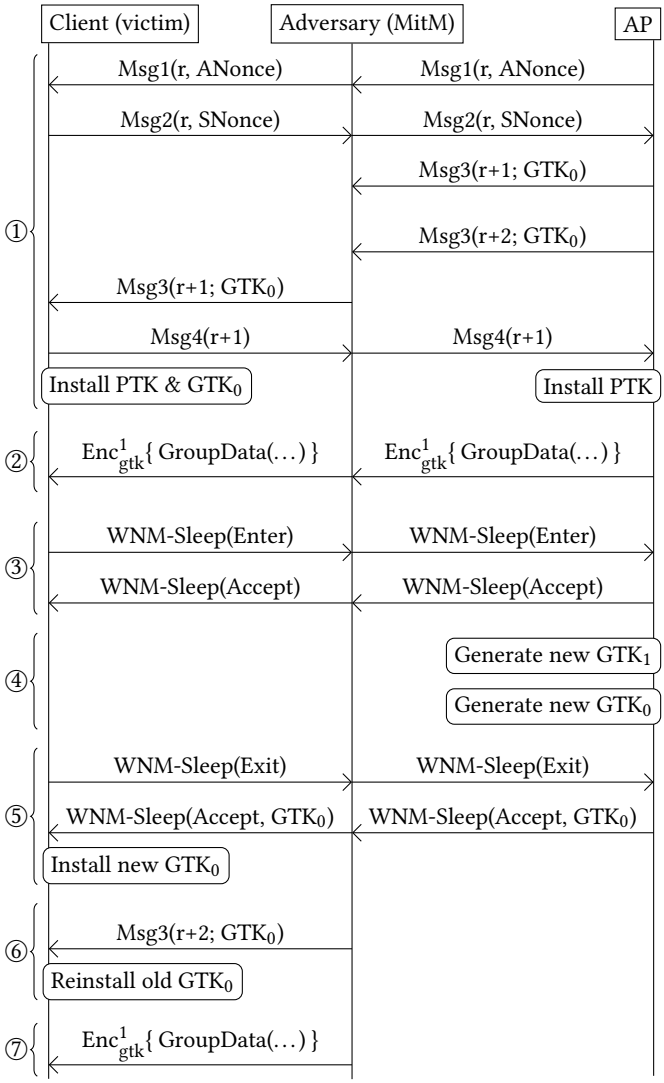

Figure 14: Abusing the interaction between the 4-way handshake and WNM-Sleep frames to bypass key reinstallation countermeasures. More precisely, we first (re)install an old (integrity) group key, after temporarily installing a different key. In this case, the reinstallation itself is caused by an EAPOL-Key frame of the 4-way handshake.