# Key Reinstallation Attacks: Breaking the WPA2 Protocol

Mathy Vanhoef
imec-DistriNet[1]
mathy.vanhoef@cs.kuleuven.be

*We introduce key reinstallation attacks. These attacks abuse features of a protocol to reinstall an already in-use key, thereby resetting nonces and/or replay counters associated to this key. We show our novel attack technique breaks several handshakes that are used in a WPA2-protected network.*

*All protected Wi-Fi networks use the 4-way handshake to generate fresh session keys. The design of this handshake was proven secure, and over its 14-year lifetime no weaknesses have been found in it. However, contrary to this history, we show that the 4-way handshake is vulnerable to key reinstallation attacks. In such an attack, the adversary tricks a victim into reinstalling an already in-use key. This is achieved by manipulating and replaying handshake messages. When the victim reinstalls the key, the associated incremental nonce and replay counter is reset to its initial value. Apart from breaking the 4-way handshake, we also show that our key reinstallation attack breaks the PeerKey, group key, and Fast BSS Transition (FT) handshake. The impact of our attacks depend on both the handshake being targeted, and the data-confidentiality protocol in use. Simplified, against AES-CCMP, an adversary can replay and decrypt packets, but cannot forge packets. Still, this makes it possible to hijack TCP streams and inject malicious data into them. Against WPA-TKIP and GCMP, the impact is catastrophic: an adversary can replay, decrypt, and forge arbitrary packets. Rather surprisingly, GCMP is especially affected because it uses the same authentication key in both communication directions.*

*Finally, we confirmed our findings in practice, and found that every Wi-Fi device is vulnerable to some variant of our attacks. Notably, our attack is exceptionally devastating against Android and Linux: it forces the client into using a predictable all-zero encryption key.*

## 1 INTRODUCTION

All protected Wi-Fi networks are secured using some version of Wi-Fi Protected Access (WPA/2). This technology relies on the 4-way handshake defined in the 802.11i

[1] This report is a summary of a paper co-authored with F. Piessens [25], who is also of imec-DistriNet.

amendment of 802.11 [3]. In this white paper, we present design flaws in this 4-way handshake, and in related handshakes. Because we target these handshakes, both WPA- and WPA2-certified products are affected by our attacks.

The 4-way handshake provides mutual authentication and session key agreement. Together with (AES)-CCMP, a data-confidentiality and integrity protocol, it forms the core of the 802.11i amendment. Since its introduction in 2003, this part of the 802.11i amendment has remained free from attacks. Its only known weaknesses are in TKIP [21, 17], which is a data-confidentiality protocol designed as a short-term solution to replace the broken WEP protocol. In other words, TKIP was never intended to be a long-term secure solution. Additionally, while several attacks against protected Wi-Fi networks were discovered over the years, these did not exploit flaws in 802.11i [26, 5, 4, 24, 14, 6]. That no major weakness has been found in CCMP and the 4-way handshake, is not surprising. After all, both have been formally proven as secure [12, 11]. With this in mind, one would assume the design of the 4-way handshake is indeed secure.

However, in spite of its history and security proofs, we show that the 4-way handshake is vulnerable to key reinstallation attacks. Moreover, we discovered similar weaknesses in other Wi-Fi handshakes.

The idea behind our attacks is rather trivial in hindsight, and can be summarized as follows. When a client joins a network, it executes the 4-way handshake to negotiate a fresh session key. It will install this key after receiving message 3 of the handshake. Once the key is installed, it will be used to encrypt normal data frames using a data-confidentiality protocol. However, because messages may be lost or dropped, the Access Point (AP) will retransmit message 3 if it did not receive an appropriate response as acknowledgment. As a result, the client may receive message 3 multiple times. Each time it receives this message, it will *reinstall* the same session key, and thereby reset the incremental transmit packet number (nonce) and receive replay counter used by the data-confidentiality protocol. We show that an attacker can force these nonce resets by collecting and replaying retransmissions of message 3. By forcing nonce reuse in this manner, the data-confidentiality protocol can be attacked, e.g., packets can be replayed, decrypted, and/or forged. The same technique is used to attack the group key, PeerKey, and fast BSS transition handshake.

When the 4-way or fast BSS transition handshake is attacked, the precise impact depends on the data-confidentiality protocol being used. If CCMP is used, arbitrary packets can be decrypted. This can be used to decrypt TCP SYN packets, and hijack TCP connections. For example, an adversary can inject malicious content into unencrypted HTTP

connections. If TKIP or GCMP is used, an adversary can both decrypt and inject arbitrary packets. Finally, when the group key handshake is attacked, an adversary can replay group-addressed frames, i.e., broadcast and multicast frames.

Our attack is especially devastating against version 2.4 and higher of `wpa_supplicant`, a Wi-Fi client commonly used on Linux. Here, the client will install an all-zero encryption key instead of reinstalling the real key. This vulnerability appears to be caused by a remark in the 802.11 standard that suggests to clear parts of the session key from memory once it has been installed [1, §12.7.6.6]. Because Android uses a modified `wpa_supplicant`, Android 6.0 and above also contain this vulnerability.

Interestingly, our attacks do not violate the formal security proofs the 4-way and group key handshake. In particular, these proofs state that the negotiated session key remains private, and that the identity of both the client and Access Point (AP) is confirmed [11]. Our attacks do not leak the session key. Additionally, an attacker cannot forge EAPOL messages and hence cannot impersonate the client or AP during (subsequent) handshakes. Instead, the problem is that the proofs do not model when a negotiated key should be installed. In practice, this means the same key can be installed multiple times, thereby resetting nonces and replay counters used by the data-confidentiality protocol.

In this report we summarize the core ideas behind our key reinstallation attack. More precisely, Section 2 covers background information, and in Section 3 we explain the attack when applied against the 4-way handshake. For additional details, we refer to our original research paper [25]. Finally, we conclude in Section 4.

# 2 BACKGROUND

In this section we introduce the various messages and handshakes used when connecting to a Wi-Fi network, and the data-confidentiality and integrity protocols of 802.11.

## 2.1 Authentication and Association

When a client wants to connect to a Wi-Fi network, it starts by (mutually) authenticating and associating with the AP. In Figure 2 this is illustrated in the association stage of the handshake. However, when first connecting to a network, no actual authentication takes places at this stage. Instead, Open System authentication is used, which allows any client to authenticate. Actual authentication will be performed during the 4-way handshake.

After (open) authentication, the client associates with the network. This is done by sending an association request to the AP. This message contains the pairwise and group cipher suites the client wishes to use. The AP replies with an association response, informing the client whether the association was successful or not.

## 2.2 The 4-way Handshake

The 4-way handshake provides mutual authentication based on a shared secret called the Pairwise Master Key (PMK), and negotiates a fresh session key called the Pairwise Transient Key (PTK). In 2005, shortly after its introduction, this handshake has been formally analyzed and proven to be secure [11]. During the 4-way handshake, the client is called the supplicant, and the AP is called the authenticator (we use these terms as synonyms). The PMK is derived from a pre-shared password in a personal network, and is negotiated using an 802.1X authentication stage in an enterprise network (see Figure 2). In turn the PTK is derived from the PMK, Authenticator Nonce (ANonce), Supplicant Nonce (SNonce), and the MAC addresses of both the supplicant and authenticator. Once generated, the PTK can be used to protect handshake messages, and to protect normal data frames using a data-confidentiality protocol. The 4-way handshake also transports the Group Temporal Key (GTK) to the supplicant.

Every message in the 4-way handshake is defined using EAPOL frames. We will briefly discuss the layout and most important fields of these frames (see Figure 1). First, the header defines which message in the handshake a particular EAPOL frame represents. We will use the notation *message n* and *MsgN* to refer to the *n*-th message of the 4-way handshake. The replay counter field is used to detect replayed frames. The authenticator always increments the replay counter after transmitting a frame. When the supplicant replies to an EAPOL frame of the authenticator, it uses the same replay counter as the one in the EAPOL frame it is responding to. The nonce field transports the random nonces that the supplicant and authenticator generate to derive a fresh session key. Next, in case the EAPOL frame transports a group key, the Receive Sequence Counter (RSC) contains the current receive replay counter (packet number) of this key. The group key itself is stored in the Key Data field, which is encrypted using the PTK. Finally, the authenticity of the full frame is protected using the PTK with a Message Integrity Check (MIC).

Figure 2 illustrates the messages that are exchanged in the 4-way handshake. We use the notation “MsgN(r, Nonce; GTK)”, which represents message N of the 4-way handshake, having a replay counter of $r$, and with the given nonce (if present). All parameters after the semicolon are stored in the key data field, and hence are encrypted using the PTK.

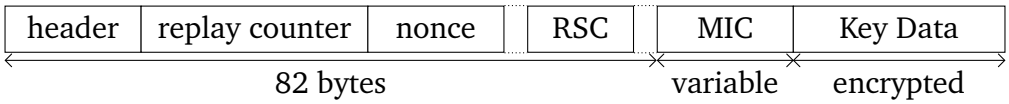


Figure 1: Simplified layout of an EAPOL frame.

The authenticator initiates the 4-way handshake by sending message 1. It contains the ANonce, and is the only EAPOL message that is not protected by a MIC. On reception of this message, the supplicant generates the SNonce and derives the PTK (also called the session key). The supplicant then sends the SNonce to the authenticator in message 2. Once the authenticator learns the SNonce, it also derives the PTK, and sends the group key (GTK) to the supplicant. Finally, to finalize the handshake, the supplicant replies with message 4 and after that installs the PTK and GTK. After receiving message 4, the authenticator also installs the PTK (the GTK is installed when the AP is started). To summarize, the first two messages are used to transport nonces, and the last two messages are used to transport the group key and to protect against downgrade attacks.

## 2.3 Confidentiality and Integrity Protocols

The 802.11i amendment contains two data-confidentiality protocols. The first is called the Temporal Key Integrity Protocol (TKIP). Nowadays TKIP is deprecated due to security concerns [27]. The second protocol is called (AES-)CCMP and is currently the most widely-used one [22]. In 2012, the 802.11ad amendment added a new data-confidentiality protocol called the Galois/Counter Mode Protocol (GCMP) [2]. Right now, 802.11ad is being rolled out under the name Wireless Gigabit (WiGig), and is expected to be adopted at a high rate over the next few years [18].

When TKIP is used, the session key (PTK) is further split into a 128-bit encryption key, and two 64-bit Message Integrity Check (MIC) keys. The first MIC key is used for AP-to-client communication, while the second key is used for the reverse direction. RC4 is used for encryption, with a unique per-packet key that is a mix of the 128-bit encryption key, the sender MAC address, and an incremental 48-bit nonce. This nonce is incremented after transmitting a frame, used as a replay counter by the receiver, and initialized to 1 when installing the TK [1, §12.5.2.6]. Message authenticity is provided by the Michael algorithm. Unfortunately, Michael is trivial to invert: given plaintext data and its MIC value, one can efficiently recover the MIC key [21, 22].

The CCMP protocol is based on the AES cipher operating in CCM mode (counter mode with CBC-MAC). It is an Authenticated Encryption with Associated Data (AEAD) algorithm, and secure as long as no Initialization Vector (IV) is repeated under a particular

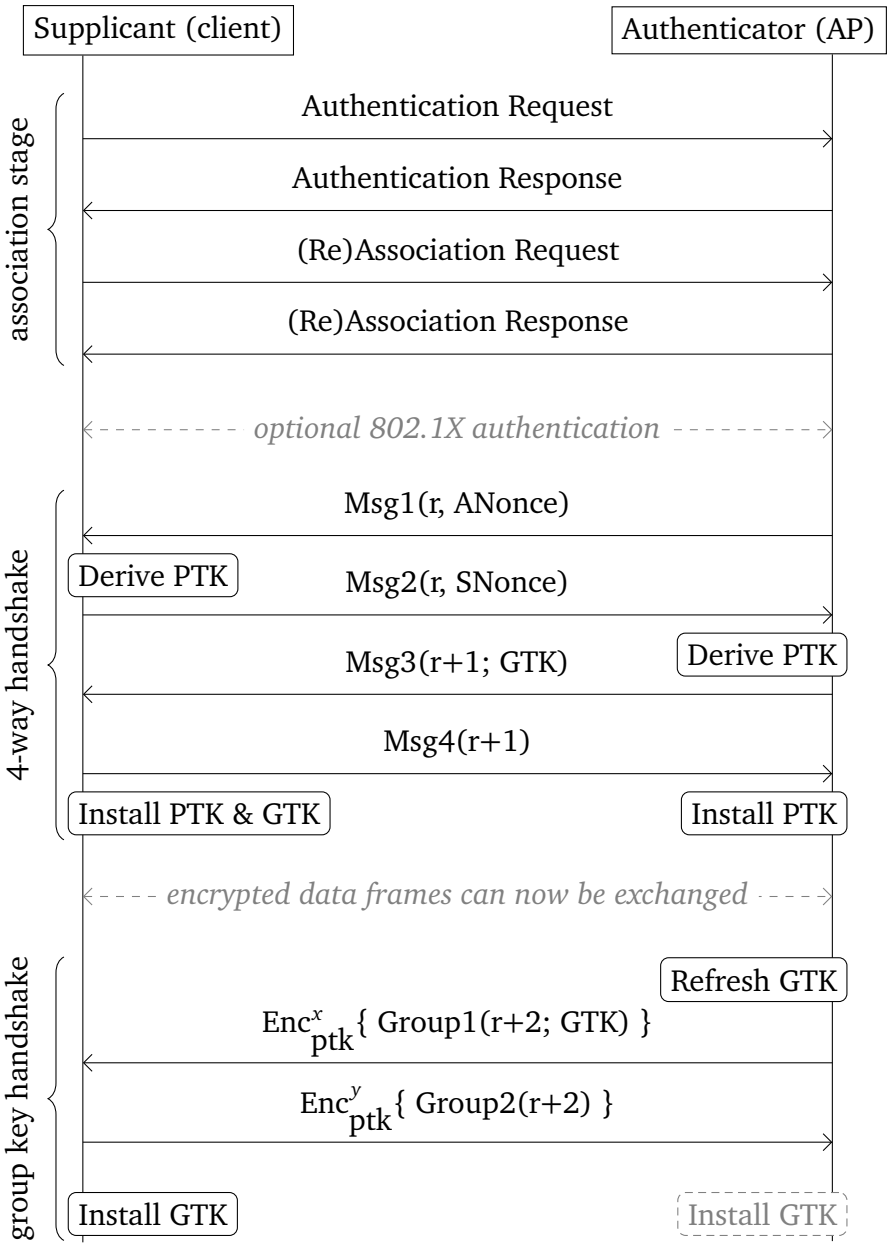


Figure 2: Messages exchanged when a supplicant (client) connects with an authenticator (AP), performs the 4-way handshake, and periodically executes the group key handshake.

key[2]. In CCMP, the IV is the concatenation of the sender MAC address, a 48-bit nonce, and some additional flags derived from the transmitted frame. The nonce is also used as a replay counter by the receiver, incremented by one before sending each frame, and initialized to 0 when installing the TK [1, §12.5.3.4.4]. This should assure that IVs do not repeat. Additionally, this construction allows the TK to be used directly as the key for both communication directions.

The GCMP protocol is based on AES-GCM, meaning it uses counter mode for encryption, with the resulting ciphertext being authenticated using the GHASH function [7]. Similar to CCMP, it is an AEAD cipher, and secure as long as no IV is repeated under a particular key. In GCMP, the IV is the concatenation of the sender MAC address and a 48-bit nonce. The nonce is also used as a replay counter by the receiver, incremented by one before sending each frame, and initialized to 0 when installing the TK [1, §12.5.5.4.4]. This should assure each IV is only used once. As with CCMP, the TK is used directly as the key for both communication directions. If a nonce is ever repeated, it is possible to reconstruct the authentication key used by the GHASH function [13].

To denote that a frame is encrypted and authenticated using a data-confidentiality protocol, we use the notation "$\text{Enc}_k^n\{\cdot\}$". Here $n$ denotes the nonce being used (and thus also the replay counter). The parameter $k$ denotes the key, which is the PTK for unicast traffic. For group-addressed traffic, i.e., broadcast and multicast frames, this is the GTK. Finally, the two notations "Data(payload)" and "GroupData(payload)" are used to represent an ordinary unicast or group-addressed data frame, respectively, with the given payload.

# 3 ATTACKING THE 4-WAY HANDSHAKE

In this section we show that the state machine behind the 4-way handshake is vulnerable to a key reinstallation attack. We then explain how to execute this attack in practice.

## 3.1 Supplicant State Machine

The 802.11i amendment (i.e., the WPA2 protocol) does not contain a formal state machine describing how the supplicant must implement the 4-way handshake. Instead, it only provides pseudo-code that describes how, but not when, certain handshake messages should be processed [3, §8.5.6]. Fortunately, 802.11r extends the 4-way handshake, and

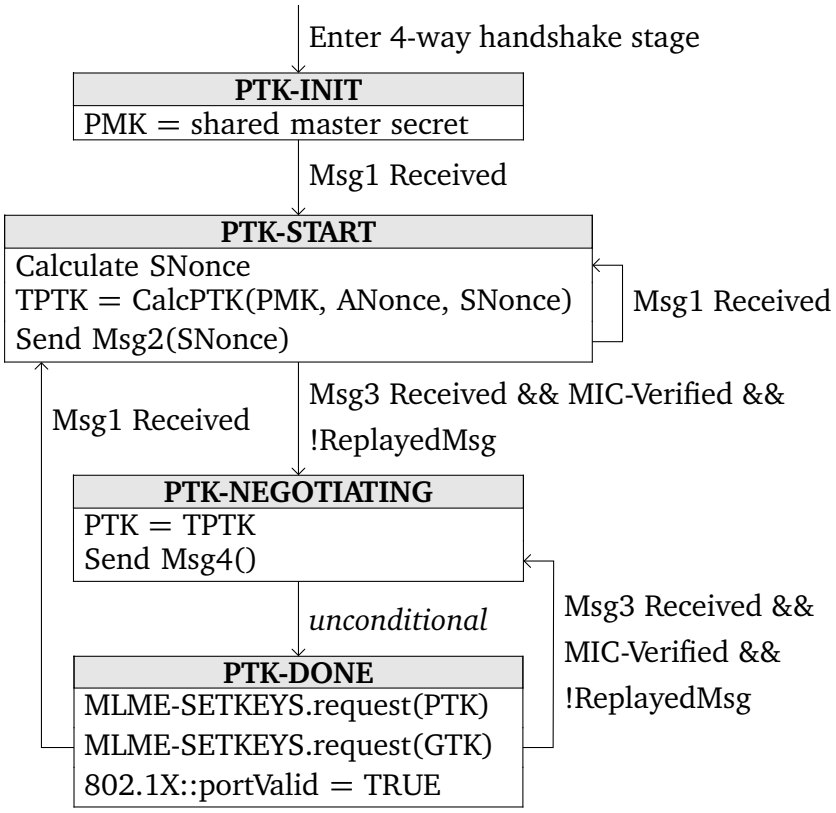


Figure 3: Client 4-way handshake state machine as defined in 802.11 [1, Fig. 13-17]. Keys are installed for usage by calling the MLME-SETKEYS.request primitive.

does provide a detailed state machine of the supplicant [1, Fig. 13-17]. Figure 3 contains a simplified description of this state machine.

When first connecting to a network and starting the 4-way handshake, the supplicant transitions to the PTK-INIT state (see Figure 3). Here, it initializes the Pairwise Master Key (PMK). When receiving message 1, it transitions to the PTK-START stage. This may happen when connecting to a network for the first time, or when the session key is being refreshed after a previous (completed) 4-way handshake. When entering PTK-START, the supplicant generates a random SNonce, calculates the Temporary PTK (TPTK), and sends its SNonce to the authenticator using message 2. The authenticator will reply with message 3, which is accepted by the supplicant if the MIC and replay counter are valid. If so, it moves to the PTK-NEGOTIATING state, where it marks the TPTK as valid by assigning it to the PTK variable, and sends message 4 to the authenticator. Then it immediately transitions to the PTK-DONE state, where the PTK and GTK are installed for usage by the data-confidentiality protocol using the MLME-SETKEYS.request primitive. Finally, it opens the 802.1X port such that the supplicant can receive and send normal data frames. Note that the state machine explicitly takes into account retransmissions of either message 1 or 3, which occur if the authenticator did not receive message 2 or 4, respectively. These retransmissions use an incremented EAPOL replay counter.

## 3.2 The Key Reinstallation Attack

Our key reinstallation attack is now easy to spot: because the supplicant still accepts retransmissions of message 3, even when it is in the PTK-DONE state, we can force a reinstallation of the PTK. More precisely, we first establish a man-in-the-middle (MitM) position between the supplicant and authenticator. We use this MitM position to trigger retransmissions of message 3 by preventing message 4 from arriving at the authenticator. As a result, it will retransmit message 3, which causes the supplicant to reinstall an already-in-use PTK. In turn, this resets the transmit nonce and receive replay counter being used by the data-confidentiality protocol. Depending on which protocol is used, this allows an adversary to replay, decrypt, and/or forge packets.

In practice, some complications arise when executing the attack. First, not all Wi-Fi clients properly implement the state machine. In particular, Windows and iOS 10 do not accept retransmissions of message 3 (see Table 1 column 2). This violates the 802.11

[2]Note that we deviate from official 802.11 terminology, where what we call the nonce is called the packet number, and what we call the IV is called the nonce.

standard. As a result, these implementations are not vulnerable to our key reinstallation attack against the 4-way handshake. Unfortunately, from a defenders perspective, both iOS and Windows are still vulnerable to our attack against the group key handshake [25].

Another minor obstacle is that we must obtain a MitM position between the client and AP. This is not possible by setting up a rogue AP with a different MAC address, and then forwarding packets between the real AP and client. Recall from Section 2.2 that the session key is based on the MAC addresses of the client and AP, meaning both would derive a different key, causing the handshake and attack to fail. Instead, we employ a channel-based MitM attack [23], where the AP is cloned on a different channel with the same MAC address as the targeted AP. This assures the client and AP derive the same session key.

The third obstacle is that certain implementations only accept frames protected using the data-confidentiality protocol once a PTK has been installed (see Table 1 column 3). This is problematic because the AP will retransmit message 3 without encryption. This means the retransmitted message 3 will be dropped by the supplicant. Although this would seem to foil our attack, techniques exists to bypass this problem [25].

In the next section, we will describe in detail how to execute our key reinstallation attack against the 4-way handshake. More precisely, we explain our attack when the client (victim) accepts plaintext retransmissions of message 3 (see Table 1 column 3). We refer to our research paper on how to execute the attack under different circumstances [25]. Table 1 column 5 summarizes which devices are vulnerable to some variant of the key reinstallation attack against the 4-way handshake [25]. We remark that the behaviour of a device depends both on the operating system, and the wireless NIC being used. For example, although Linux accepts plaintext retransmissions of message 3, the Wi-Fi NICs used in several Android devices reject them. However, Android phones with a different wireless NIC may in fact accept plaintext retransmissions of message 3.

## 3.3 Plaintext Retransmission of message 3

If the victim still accepts plaintext retransmissions of message 3 after installing the session key, our key reinstallation attack is straightforward. First, the adversary uses a channel-based MitM attack so they can manipulate handshake messages [23]. Then they block message 4 from arriving at the authenticator. This is illustrated in stage 1 of Figure 4. Immediately after sending message 4, the victim will install the PTK and GTK. At this point the victim also opens the 802.1X port, and starts transmitting normal data

Table 1: Behaviour of clients: 2nd column shows whether retransmission of message 3 are accepted, 3rd whether plaintext EAPOL messages are accepted if a PTK is configured, 4th whether it accepts plaintext EAPOL messages if sent immediately after the first message 3. The last two columns denote if the client is vulnerable to a key reinstallation attack against the 4-way or group key handshake, respectively.

| Implementation | Re. Msg3 | Pt. EAPOL | Quick Pt. | 4-way | Group |
|---|---|---|---|---|---|
| OS X 10.9.5 | ✓ | ✗ | ✗ | ✓ | ✓ |
| macOS Sierra 10.12 | ✓ | ✗ | ✗ | ✓ | ✓ |
| iOS 10.3.1 [b] | ✗ | N/A | N/A | ✗ | ✓ |
| `wpa_supplicant` v2.3 | ✓ | ✓ | ✓ | ✓ | ✓ |
| `wpa_supplicant` v2.4+ | ✓ | ✓ | ✓ | ✓[a] | ✓ |
| Android 6.0.1 | ✓ | ✗ | ✓ | ✓[a] | ✓ |
| OpenBSD 6.1 (rum) | ✓ | ✗ | ✗ | ✗ | ✓ |
| OpenBSD 6.1 (iwn) | ✓ | ✗ | ✗ | ✓ | ✓ |
| Windows 7 [b] | ✗ | N/A | N/A | ✗ | ✓ |
| Windows 10 [b] | ✗ | N/A | N/A | ✗ | ✓ |
| MediaTek | ✓ | ✓ | ✓ | ✓ | ✓ |

[a] Due to a bug, an all-zero key will be installed, see Section 3.4.

[b] Certain tests are irrelevant (not applicable) because the implementation does not accept retransmissions of message 3.

frames (recall Section 2.2). Notice that the first data frame uses a nonce value of 1 in the data-confidentiality protocol. Then, in the third stage of the attack, the authenticator retransmits message 3 because it did not receive message 4. The adversary forwards the retransmitted message 3 to the victim, causing it to reinstall the PTK and GTK. As a result, it resets the nonce and replay counter used by the data-confidentiality protocol. Note that the adversary cannot replay an old message 3, because its EAPOL replay counter is no longer fresh. We ignore stage 4 of the attack for now. Finally, when the victim transmits its next data frame, the data-confidentiality protocol reuses nonces. Note that an adversary can wait an arbitrary amount of time before forwarding the retransmitted message 3 to the victim. Therefore, we can control the amount of nonces that will be reused. Moreover, an adversary can always perform the attack again by deauthenticating the client, after which it will reconnect with the network and execute a new 4-way handshake.

Figure 4 also shows that our key reinstallation attack occurs spontaneously if message 4 is lost due to background noise. Put differently, clients that accept plaintext retransmissions of message 3, may already be reusing nonces without an adversary even being present. Inspired by this observation, an adversary could also selectively jam message 4 [23], resulting in a stealthy attack indistinguishable from random background interference.

We now return to stage 4 of the attack. The goal of this stage is to complete the handshake at the authenticator side. This is not trivial because the victim already installed the PTK, meaning its last message 4 is encrypted.[3] And since the authenticator did not yet install the PTK, it will normally reject this encrypted message 4.[4] However, a careful inspection of the 802.11 standard reveals that the authenticator may accept *any* replay counter that was used in the 4-way handshake, not only the latest one [1, §12.7.6.5]:

> "On reception of message 4, the Authenticator verifies that the Key Replay Counter field value is one that it used on this 4-way handshake."

In practice, we found that several APs indeed accept an older replay counter. More precisely, some APs accept replay counters that were used in a message to the client, but were not yet used in a reply from the client. These APs will accept the older unencrypted message 4, which has the replay counter $r + 1$ in Figure 4. As a result, these APs will install the PTK, and will start sending encrypted unicast data frames to the client.

[3] The 802.11 standard says that a retransmitted message 4 must be sent in plaintext in the initial 4-way handshake [1, §12.7.6.5], but nearly all clients send it using encryption.

[4] Similar to fixes in [19, 20], a non-standard implementation may already install the PTK for reception-only after sending message 3. We found no AP doing this though.

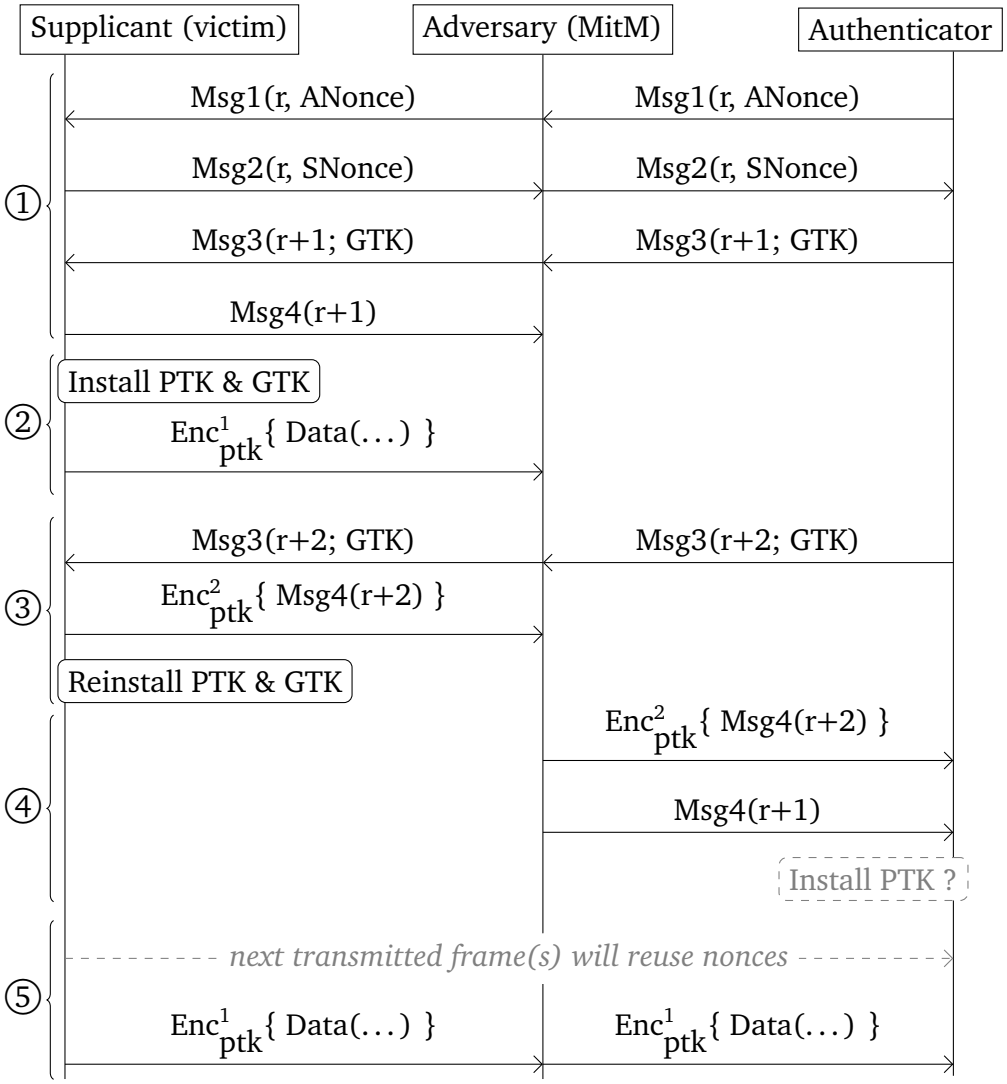


Figure 4: Key reinstallation attack against the 4-way handshake, when the supplicant (victim) still accepts plaintext retransmissions of message 3 if a PTK is installed.

Although Figure 4 only illustrates nonce reuse in frames sent by the client, our attack also enables us to replay frames. First, after the client reinstalls the GTK in stage 3 of the attack, broadcast and multicast frames that the AP sent after retransmitting message 3 can be replayed. This is because replay counters are also reset when reinstalling a key. Second, if we can make the AP install the PTK, we can also replay unicast frames sent from the AP to the client.

We confirmed that the attack shown in Figure 4 works against MediaTek's implementation of the Wi-Fi client, and against `wpa_supplicant` (see also Section 3.4). How we attacked other implementations is explained in our research paper [25].

## 3.4 Linux and Android all-zero key (re)installation

Our key reinstallation attack against the 4-way handshake uncovered special behavior in `wpa_supplicant`. First, version 2.3 and lower are vulnerable to our attacks without unexpected side-effects. However, we found that version 2.4 and 2.5 install an all-zero encryption key when receiving a retransmitted message 3. This vulnerability appears to be caused by a remark in the 802.11 standard that indirectly suggests to clear the encryption key from memory once it has been installed [1, §12.7.6.6]. Version 2.6 fixed this bug by only installing the TK when receiving message 3 for the first time [15]. However, when patching this bug, only a benign scenario was considered where message 3 got retransmitted because message 4 was lost due to background noise. They did not consider that an active attacker can abuse this bug to force the installation of an all-zero key. As a result, the patch was not treated as security critical, and was not backported to older versions. Moreover, an attacker can still trick `wpa_supplicant` 2.6 into installing an all-zero key, by injecting a forged message 1 between the original and retransmitted message 3 [16].

Because Android internally uses a slightly modified version of `wpa_supplicant`, it is also affected by these implementation-specific vulnerabilities. In particular, we inspected the official source code repository of Android's `wpa_supplicant` [10, 8], and found that Android version 6.0 and above contain the all-zero encryption key vulnerability. Though third party manufacturers might use a different `wpa_supplicant` version in their Android builds, this is a strong indication that all Android 6.0 devices and newer are vulnerable. In other words, 50% of Android smartphones are vulnerable to the all-zero encryption key vulnerability [9]. Finally, we also empirically confirmed that Chromium is vulnerable to the all-zero encryption key vulnerability.

# 4 CONCLUSION

Despite a security proof of both the 4-way handshake and the CCMP encryption protocol, we showed that when combined, they are vulnerable to key reinstallation attacks. One reason this attack went unnoticed for so long, is that the two individual security proofs were never combined. Note that our attacks do not violate the security properties of each individual security proof. Instead, the problem is that the models in the proofs do not specify when a key should be installed for usage by the data-confidentiality protocol.

All Wi-Fi clients we tested were vulnerable to our attack against the group key handshake. This enables an adversary to replay broadcast and multicast frames. When the 4-way or fast BSS transition handshake is attacked, the precise impact depends on the data-confidentiality protocol being used. In all cases though, it is possible to decrypt frames and thus hijack TCP connections. This enables the injection of data into unencrypted HTTP connections. Moreover, against recent version of Linux and Android, our attack causes the installation of an all-zero key. This completely voids any security guarantees.

# REFERENCES

[1] IEEE Std 802.11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Spec*, 2016.

[2] IEEE Std 802.11ad. *Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band*, 2012.

[3] IEEE Std 802.11i. *Amendment 6: Medium Access Control (MAC) Security Enhancements*, 2004.

[4] Gal Beniamini. Over the air: Exploiting Broadcom's Wi-Fi stack, 2017.

[5] Laurent Butti and Julien Tinnes. Discovering and exploiting 802.11 wireless driver vulnerabilities. *Journal in Computer Virology*, 4(1):25–37, 2008.

[6] Aldo Cassola, William Robertson, Engin Kirda, and Guevara Noubir. A practical, targeted, and stealthy attack against WPA enterprise authentication. In *NDSS Symp.*, April 2013.

[7] Morris Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) for confidentiality and authentication. In *NIST Special Publication 800-38D*, November 2007.

[8] Google. Codenames, tags, and build numbers, 2017.

[9] Google. Dashboards: Platform versions, May 2017.

[10] Google Git. wpa supplicant 8, 2017.

[11] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *CCS*, 2005.

[12] Jakob Jonsson. On the security of CTR+ CBC-MAC. In *SAC*, 2002.

[13] Antoine Joux. 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.

[14] Eduardo Novella Lorente, Carlo Meijer, and Roel Verdult. Scrutinizing WPA2 password generating algorithms in wireless routers. In *USENIX WOOT*, 2015.

[15] Jouni Malinen. Fix TK configuration to the driver in EAPOL-Key 3/4 retry case. Hostap commit `ad00d64e7d88`, October 2015.

[16] Jouni Malinen. Wpa packet number reuse with replayed messages and key reinstallation, 2016.

[17] Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. Plaintext recovery attacks against WPA/TKIP. In *FSE*, 2014.

[18] Grand View Research. Wireless gigabit (wigig) market size to reach $7.42 billion by 2024, 2017.

[19] Robert Stacey, Adrian Stephens, Jesse Walker, Herbert Liondas, and Emily Qi. Rekeying protocol fix, 2010.

[20] Robert Stacey, Adrian Stephens, Jesse Walker, Herbert Liondas, and Emily Qi. Rekeying protocol fix text, 2010.

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

[22] Mathy Vanhoef and Frank Piessens. Practical verification of WPA-TKIP vulnerabilities. In *ASIA CCS*, pages 427–436. ACM, 2013.

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

[24] Mathy Vanhoef and Frank Piessens. Predicting, decrypting, and abusing WPA2/802.11 group keys. In *USENIX Security*, 2016.

[25] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In *CCS*. ACM, 2017.

[26] Stefan Viehböck. Brute forcing Wi-Fi protected setup, 2011.

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