Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2

Anonymous Author(s)

ABSTRACT

This paper introduces key reinstallation attacks. These attacks abuse features of a protocol to reinstall a key that is already in use, and thereby reset nonces and/or replay counters associated to this key.

All encrypted 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 show that our key reinstallation attack also 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 6.0: it forces the client into using a predictable all-zero encryption key.

CCS CONCEPTS

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

KEYWORDS

handshake, session key, nonce, replay counter, reuse

1 INTRODUCTION

All encrypted Wi-Fi networks are secured using some version of Wi-Fi Protected Access (WPA/2). Moreover, nowadays even public hotspots are able to use authenticated encryption thanks to the Hotspot 2.0 program [7]. All these technologies rely on the 4-way handshake defined in the 802.11i amendment of 802.11 [4]. In this work, we present design flaws in the 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 foundation of the 802.11i amendment. Since its first introduction in 2003, under the name WPA, this core part of the 802.11i amendment has remained free from attacks. Indeed, the only currently known weaknesses of 802.11i are in (WPA-)TKIP [49, 55]. This data-confidentiality protocol was designed as a short-term solution to the broken WEP protocol. In other words, TKIP was never intended to be a long-term secure solution. Additionally, while several attacks against encrypted Wi-Fi networks were discovered over the years, these did not exploit flaws in 802.11i. Instead, attacks exploited flaws in Wi-Fi Protected Setup (WPS) [59], flawed drivers [12, 17], predictable pre-shared keys [37], insecure enterprise authentication [18], and so on. 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 [32, 34]. With this in mind, one might reasonably assume the design of the 4-way handshake is indeed secure.

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. That is, we also attack the PeerKey handshake, the group key handshake, and the Fast BSS Transition (FT) handshake.

The core 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 nonce and 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 none 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. In turn, 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. Although GCMP is a relatively new addition to Wi-Fi, it is expected to be adopted at a high rate in the next few years [50]. Finally, when the group key handshake is attacked, an adversary can replay broadcast and multicast frames.

Our attack is especially devastating against version 2.4 and 2.5 of `wpa_supplicant`, a Wi-Fi client commonly used on Linux. Here, the client will install an all-zero data-confidentiality 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 Android Wear 2.0 also contain this vulnerability. As a result, currently 31.2% of Android devices are vulnerable to this exceptionally devastating variant of our attack [27].

Interestingly, our attacks do not violate the security properties proven in formal analysis of 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 [32]. Our attacks do not leak the session key. Additionally, although normal data frames can be forged if TKIP or GCMP is used, an attacker cannot forge EAPOL messages and hence cannot impersonate the supplicant or authenticator during (subsequent) handshakes. Instead, the problem is that the proofs do not model key installation. Put differently, their models do not state 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.

To summarize, our main contributions are:

- We introduce key reinstallation attacks. Here, an attacker forces the reinstallation of an already in-use key, thereby resetting any associated nonces and/or replay counters.
- We show that the 4-way handshake, PeerKey handshake, group key handshake, and fast BSS transition handshake are vulnerable to key reinstallation attacks.
- We devise attack techniques to carry out our attacks in practice. This demonstrates that all implementations are vulnerable to some variant of our attack.
- We evaluate the practical impact of nonce reuse for all data-confidentiality protocols of 802.11.

The remainder of this paper is structured as follows. Section 2 introduces relevant aspects of the 802.11 standard. Our key reinstallation attack is illustrated against the 4-way and PeerKey handshake in Section 3, against the group key handshake in Section 4, and against the fast BSS transition handshake in Section 5. In Section 6 we more deeply asses the impact of our attacks, present countermeasures, and discuss where the security proofs failed. Finally, we present related work in Section 7 and conclude in Section 8.

2 BACKGROUND

In this section we introduce the 802.11i amendment, 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 The 802.11i Amendment

After researchers showed that Wired Equivalent Privacy (WEP) was fundamentally broken [25, 54], the IEEE offered a more robust solution in the 802.11i amendment of 802.11. This amendment defines the 4-way handshake (see Section 2.3), and two data-confidentiality and integrity protocols called (WPA-)TKIP and (AES-)CCMP (see Section 2.4). While the 802.11i amendment was under development, the Wi-Fi Alliance already began certifying devices based on draft version D3.0 of 802.11i. This certification program was called Wi-Fi Protected Access (WPA). Once the final version D9.0 of 802.11i was ratified, the WPA2 certification was created based on this officially ratified version. Because both WPA and WPA2 are based on 802.11i, they are almost identical on a technical level. The main difference is that WPA2 mandates support for the more secure CCMP, and optionally allows TKIP, while the reverse is true for WPA.

Required functionality of both WPA and WPA2, and used by all encrypted Wi-Fi networks, is the 4-way handshake. Even enterprise networks, and personal networks, rely on the 4-way handshake. Hence, all encrypted Wi-Fi networks are affected by our attacks.

The 4-way handshake, group key handshake, and CCMP protocol, have formally been analyzed and proven to be secure [32, 34].

2.2 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. Real authentication is only done at this stage, when roaming between two APs of the same network, using the fast BSS transition handshake (see Section 3).

After this, the supplicant 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 supplicant whether the association was successful or not.

2.3 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). During this 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 negotiated using an 802.1x authentication stage in an enterprise network (see Figure 2). The PTK is derived from the Authenticator Nonce (ANonce), Supplicant Nonce (SNonce), and the MAC addresses of both the supplicant and authenticator. It is split into a Key Confirmation Key (KCK), Key Encryption Key (KEK), and Temporal Key (TK). The KCK and KEK are used to protect handshake messages, while the TK is used to protect normal data frames with a data-confidentiality protocol. The 4-way handshake also transports the current Group Temporal Key (GTK) to the supplicant.

Every message in the 4-way handshake is defined using EAPOL frames. Figure 1 illustrates the layout of these frames. We will briefly discuss its most important fields. 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 received EAPOL frame. The nonce field transports the random nones that the supplicant and authenticator generate to derive a fresh session key. Next, the Receive Sequence Counter (RSC) is only used by Message 3 in the 4-way handshake, and contains the replay counter of the group key being transported. The group key itself is stored in the Key Data field,

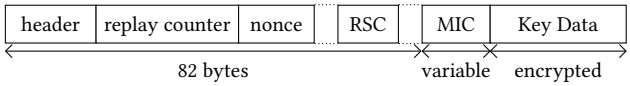

Figure 1: Simplified layout of an EAPOL frame.

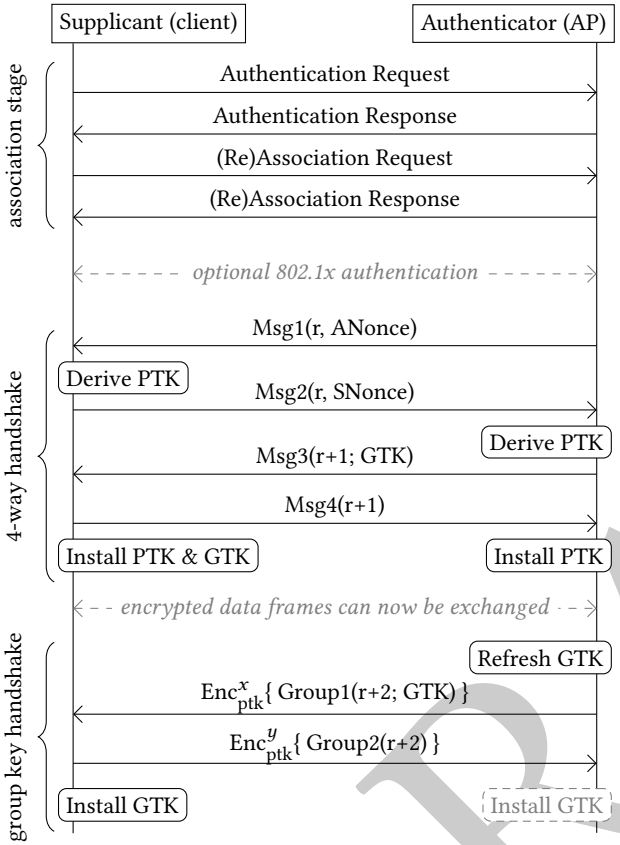

Figure 2: Messages exchanged when a supplicant (client) connects with an authenticator (AP), performs the 4-way handshake, and periodically executes the group key handshake.

which is encrypted using the KEK. Finally, the integrity of the frame is protected using the KCK with a Message Integrity Check (MIC).

Figure 2 illustrates the 4-way handshake. In it, we use the following notation:

MsgN(r, Nonce; GTK)

It 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 KEK (recall Figure 1).

The authenticator initiates the 4-way handshake by sending Message 1. It contains the ANonce, and is the only EAPOL messages that is not protected by a MIC. On reception of this message, the supplicant generates the SNonce and derives the PTK (i.e., 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 this message, the authenticator also installs the PTK (the GTK is installed when the AP is started). In essence, 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.

Note that in an existing connection, the PTK can be refreshed by initiating a new 4-way handshake. During this rekey, all 4-way handshake messages are encrypted by the data-confidentiality protocol using the current PTK (we rely on this in Section 3.4).

2.4 Confidentiality and Integrity Protocols

The 802.11i amendment defines two data-confidentiality protocols. The first is called the Temporal Key Integrity Protocol (TKIP). However, nowadays TKIP is deprecated due to security concerns [61]. The second protocol is commonly called (AES-)CCMP, and is currently the most widely-used data-confidentiality protocol [57]. In 2012, the 802.11ad amendment added a new data-confidentiality protocol called the Galios/Counter Mode Protocol (GCMP) [3]. This amendment also adds support for short-range communications in the 60 GHz band, which requires a fast cipher such as GCM [3]. 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 [50]. Finally, the 802.11ac amendment further extends GCMP by adding support for 256-bit keys [2].

When TKIP is used, the Temporal Key (TK) part of 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 communications, while the second key is used for the reverse direction. Encryption is done using RC4, 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 [55].

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 key[1]. 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 is supposed to 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 [23]. 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

[1]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.

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 normally assures 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 [35].

To denote that a frame is encrypted and authenticated using a data-confidentiality protocol, we use the following 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 (session key) for unicast traffic, and the GTK (group key) for multicast and broadcast traffic. Finally, the two notations

Data(payload)
BroadcastData(payload)

are used to represent an ordinary unicast or broadcast data frame, respectively, with the given payload.

2.5 The Group Key Handshake

When the authenticator refreshes the group key, it executes the group key handshake to distribute the new key to all clients. This handshake was proven to be secure in [32], and is shown in the last stage of Figure 3. The authenticator initiates the handshake by sending a Group Message 1 to all clients. The supplicant acknowledges the receipt of the new group key by replying with Group Message 2. Depending on the implementation, the authenticator installs the GTK either after sending Group Message 1, or after receiving a reply from all connected clients (see Section 4). Note that Group Message 1 also contains the current replay counter value of the group key in the RSC field (recall Figure 1).

Both messages in the group key handshake are defined using EAPOL frames, and are represented using Group1 and Group2 in Figure 2. Note that Group Message 1 stores the new group key in the Key Data field, and hence is encrypted using the KEK (recall Figure 1). Since at this point a PTK is installed, the complete EAPOL frame is also protected using a data-confidentiality protocol.

Finally, if a client transmits a broadcast or multicast frame, she first sends it as a unicast frame to the AP. The AP then encrypts the frame using the group key, and broadcasts it to all clients. This assures all clients within the range of the AP receive the frame.

3 ATTACKING THE 4-WAY HANDSHAKE

In this section we show that the state machine behind the 4-way handshake is vulnerable to key reinstallation attacks. We then demonstrate how to execute our attacks in real-life environments.

3.1 Supplicant State Machine

The 802.11i amendment 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 [4, §8.5.6].[2]

[2]Strangely, this pseudo-code is only present in the original 802.11i amendment. Later revisions of the 802.11 standard, which are supposed to combine all existing amendments into one updated document, no longer contain this pseudo-code.

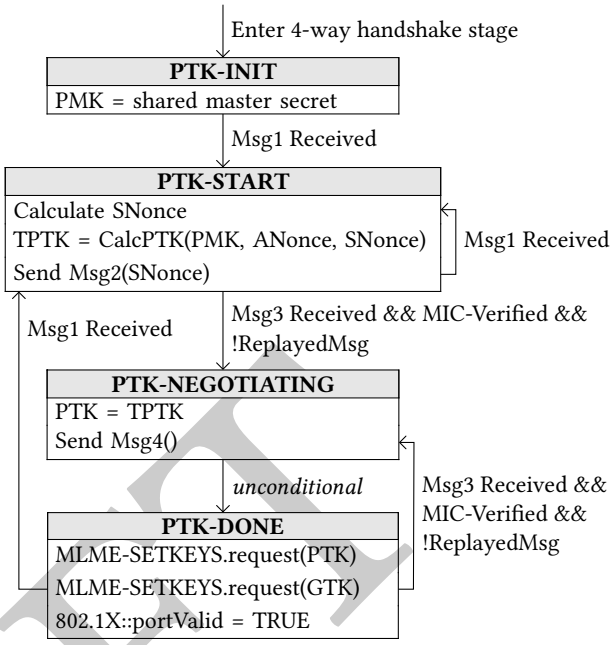

Figure 3: Supplicant 4-way handshake state machine as defined in the 802.11 standard [1, Fig. 13-17]. Keys are installed for usage by calling the MLME-SETKEYS.request primitive.

Fortunately, 802.11r slightly extends the 4-way handshake, and does provide a detailed state machine of the supplicant [1, Fig. 13-17]. Figure 2 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 2). 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.

We confirmed that the state machine in 802.11r matches the original state machine, that was "defined" in 802.11i using textual descriptions scattered throughout the amendment. Most importantly, we verified two properties which we abuse in our key reinstallation attack. First, 802.11i states that the AP retransmits Message 1 or 3 if

it did not receive a reply [4, §8.5.3.5]. Therefore, the client must handle retransmissions of Message 1 or 3, matching the state machine of 802.11r. Additionally, 802.11i states that the client should install the PTK after processing and replying to message 3 [4, §8.5.3.3]. This again matches the state machine given in 802.11r.

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 nonce 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 Section 6.1 we will explore in detail what the practical impacts of nonce reuse are for each data-confidentiality protocol.

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 do not accept retransmissions of Message 3 (see Table 1 column 2). This violates the 802.11 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 (see Section 4).

A second minor obstacle is that we must obtain a MitM position between the client and AP. This is not possible by setting up a rouge AP with a different MAC address, and forwarding packets between the real AP and client. Recall from Section 2.3 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 [58], 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 for our attack, because the authenticator will retransmit Message 3 without encryption by the data-confidentiality protocol. This means the retransmitted message will be ignored by the supplicant. Although this appears to foil our attack at first sight, we found a technique to bypass this problem (see Section 3.4).

In the next two Sections, we will describe in detail how to execute our key reinstallation attack against the 4-way handshake. First when the client (victim) accepts plaintext retransmissions of Message 3 (see Table 1 column 3), then when the victim only accepts encrypted retransmissions of Message 3 (see Table 1 column 4). Table 1 column 6 summarizes which devices are vulnerable to some variant of key reinstallation attack against the 4-way handshake.

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, and 5th whether it is affected by the attack of Section 3.4. 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.	Quick Ct.	4-way	Group
OS X 10.9.5	✓	✗	✗	✓	✓	✓
macOS Sierra 10.12.4	✓	✗	✗	✓	✓	✓
iOS 10.3.1 [c]	✗	N/A	N/A	N/A	✗	✓
`wpa_supplicant` v2.3	✓	✓	✓	✓	✓	✓
`wpa_supplicant` v2.4-5	✓	✓	✓	✓[a]	✓[a]	✓
`wpa_supplicant` v2.6	✓	✓	✓	✓[b]	✓[b]	✓
Android 6.0.1 (i9305)	✓	✗	✓	✓	✓	✓
OpenBSD 6.1 (rum)	✓	✗	✗	✗	✗	✓
OpenBSD 6.1 (iwn)	✓	✗	✗	✓	✓	✓
Windows 7 [c]	✗	N/A	N/A	N/A	✗	✓
Windows 10 [c]	✗	N/A	N/A	N/A	✗	✓
MediaTek	✓	✓	✓	✓	✓	✓

[a] Due to a bug, an all-zero key will be installed, see Section 6.3.
[b] Only the group key is reinstalled in the 4-way handshake.
[c] Certain tests are irrelevant (not applicable) because the implementation does not accept 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 she can manipulate handshake messages [58]. Then she blocks 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 frames (recall Section 2.3). 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 forward 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.

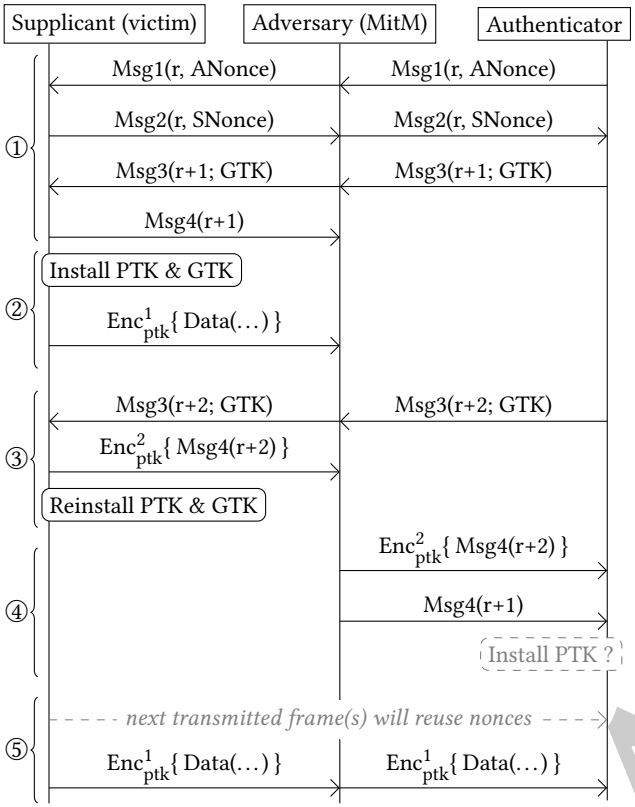

Figure 4: Key reinstallation attack against the 4-way handshake, when the supplicant (victim) still accepts plaintext retransmissions of Message 3 if a PTK is installed.

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 [58], resulting in a stealthy attack that is 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 (see column 2 in Table 2). These APs will accept the older unencrypted Message 4, which has the replay counter $r + 1$ in Figure 4. As a result, these AP 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]The 802.11 standard allows the AP to already install the PTK for reception-only after sending Message 3 [1, §12.7.6.4]. However, we found no device doing this in practice.

Table 2: Behaviour of Access Points. The 2nd column shows whether it accepts replay counters it used in a message to the client, but did not yet receive in a reply, or if it only accepts the latest used counter. Column 3 shows whether the GTK is installed immediately after sending Group Message 1, or if this is delayed until all clients replied with Group Message 2.

Implementation	Replay Check	GTK Install Time
802.11 standard	not yet received	delayed
Broadcom	not yet received	immediate
Hostapd	not yet received	delayed
OpenBSD	latest only	delayed
MediaTek	latest only	immediate
Aironet (Cisco)	latest only	immediate
Aerohive	not yet received	delayed
Ubiquiti	not yet received	delayed
Windows 7	latest only	*The group key is never refreshed*
Windows 10	latest only	
Apple OSes	latest only	

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 certain versions of `wpa_supplicant` (see also Section 6.3). How other implementations can be attack is explained in the next section.

3.4 Encrypted Retransmission of Message 3

We now describe how we can attack clients that, once they installed the PTK, only accept encrypted retransmissions of Message 3. To accomplish this, we exploit an inherent race condition between the entity executing the 4-way handshake, and the entity implementing the data-confidentiality protocol.

As a warm-up, we first attack Android's implementation of the supplicant. Here, we found that Android accepts plaintext retransmissions of Message 3 when they are sent immediately after the original Message 3 (see column 4 of Table 1). Figure 5 shows why this happens, and how it can be exploited. Note that the AP is not drawn in this figure: its actions are clear from context. In our attack, we first let the client and AP exchange Message 1 and 2. However, we do not forward the first Message 3 to the client. Instead we wait until the AP retransmits a second Message 3. In stage two of the attack, we send both Message 3's instantly after one another to the client. The wireless NIC, which implements the data-confidentiality protocol, does not have a PTK installed, and hence forwards both messages to the packet receive queue of the main CPU. The main

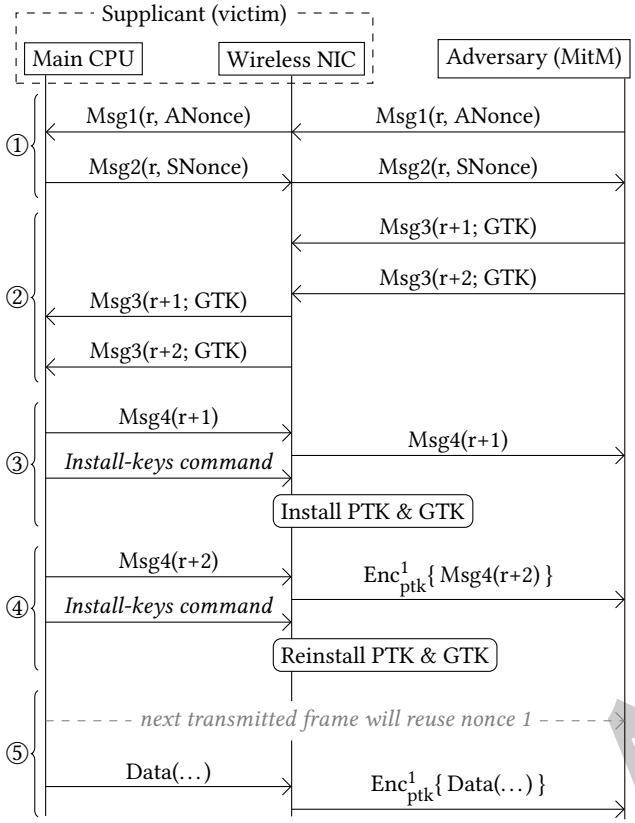

Figure 5: Key reinstallation attack against the 4-way handshake, when the victim accepts a plaintext Message 3 retransmission if sent instantly after the first one. We assume encryption and decryption is offloaded to the wireless NIC.

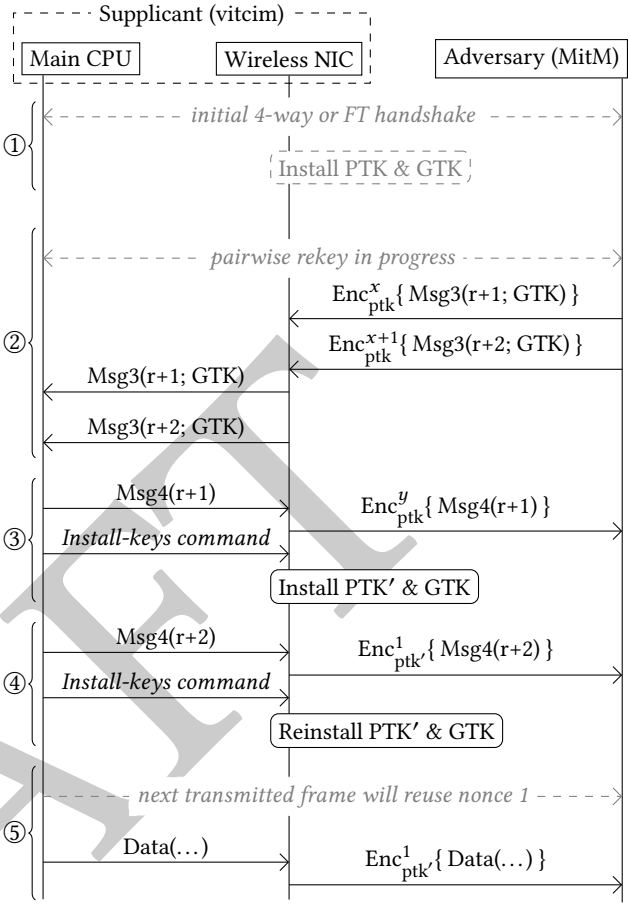

Figure 6: Key reinstallation attack against the 4-way handshake, when the victim only accepts encrypted Message 3 retransmissions once a PTK is installed. We assume encryption and decryption is offloaded to the wireless NIC.

CPU, which implements the 4-way handshake, replies to the first Message 3 and commands the wireless NIC to install the PTK. In stage 4 of the attack, the main CPU of the client grabs the second Message 3 from its receive queue. Although it notices the frame was not encrypted, Android and Linux allow unencrypted EAPOL frames as an exception, and therefore the main CPU will process the retransmitted Message 3. Because the NIC has just installed the PTK, the reply will be encrypted with a nonce value of 1. After this, it commands the wireless NIC to reinstall the PTK. By doing this, the NIC resets the nonce and replay counter associated to the PTK, meaning the next transmitted data frame will reuse nonce 1.

We now show how to attack OpenBSD, OS X, and macOS (see Table 1 column 5). These devices only accept encrypted retransmissions of Message 3. Similar to the Android attack, we abuse race conditions between the wireless NIC and main CPU. However, we now target a 4-way handshake execution that refreshes (rekeys) the PTK. Recall from Section 2.3 that all messages transmitted during a rekey undergo encryption by the data-confidentiality protocol.

Figure 6 illustrates the details of the attack. Note that the AP is not draw in this figure: its actions are clear from context. Again the adversary uses a channel-based MitM position [58]. She then lets the victim and adversary execute the initial 4-way handshake, and waits until a second 4-way handshake is initiated to refresh the PTK. Even though she only sees encrypted frames, messages in the 4-way handshake can be detected by their unique length and destination. At this point, the attack is analogous to the Android case. That is, in stage 2 of the attack, the adversary does not instantly forward the first Message 3. Instead, she waits until the AP retransmits Message 3, and then forwards both messages right after one another to the victim (see Figure 6 stage 2). The wireless NIC will decrypt both messages using the current PTK, and forwards them to the packet receive queue of the main CPU. In the third stage of the attack, the main CPU of the victim processes the first Message 3, replies to it, and commands the NIC to install the new PTK. In the fourth stage, the main CPU picks the second Message 3 from the receive queue. Since a PTK is installed, OpenBSD, OS X, and macOS (here called the main CPU) will mandate that the message was encrypted. However, they do not check under which key the message was encrypted. As a result, even though the message was decrypted under the old PTK, the main CPU will process it. The Message 4 sent as a reply is now encrypted under the new PTK

using a nonce value of 1. After this, the main CPU commands the NIC to reinstall the PTK, thereby resetting the nonce and replay counters. Finally, the next data frame that the victim transmits will again be encrypted using the new PTK with a nonce of 1. We confirmed this attack against OpenBSD 6.1, OS X 10.9.5, and macOS Sierra 10.12.4.

OpenBSD is only vulnerable if encryption is offloaded to the wireless NIC. For example, the `iwn` driver and associated devices support hardware encryption, and therefore are vulnerable. However, the `rum` driver performs software encryption in the same entity as the 4-way handshake, and is not vulnerable (see Table 1 column 5).

This attack technique requires us to wait until a rekey of the session key occurs. Several APs do this every hour [55], some examples being [19, 21]. In practice, clients can also request a rekey by sending an EAPOL frame to the AP with the `Request` and `Pairwise` bits set. Amusingly, Broadcom routers do not verify the authenticity (MIC) of this frame, meaning an adversary can force Broadcom APs into starting a rekey handshake. All combined, we can assume a rekey will eventually occur, meaning an adversary can carry out the key reinstallation attack.

3.5 Attacking the PeerKey Handshake

The PeerKey handshake is related to the 4-way handshake, and used when two clients want to communicate with each other directly in a secure manner. It consists of two phases [1, §12.7.8]. In the first phase, a Station-To-Station Link (STSL) Master Key (SMK) handshake is performed. It negotiates a shared master secret between both clients. In the second phase, fresh session keys are derived from this master key using the STSL Transient Key (STK) handshake. Although this protocol does not appear to be widely supported [41], it forms a good test case to gauge how applicable our key reinstallation technique is.

Unsurprisingly, the SKM handshake is not affected by our key reinstallation attack. After all, the master key negotiated in this handshake is not used by a data-confidentiality protocol, meaning there are no nonces or replay counters to reset. However, the STK handshake is based on the 4-way handshake, and it does install a key for use by a data-confidentiality protocol. As a result, it can be attacked in precisely the same manner as the 4-way handshake. We confirmed that the PeerKey handshake can be attacked in practice using two instances of `wpa_suppcliant`. We did not find other devices that support PeerKey. As a result, the impact of our attack against the PeerKey handshake is rather low.

4 BREAKING THE GROUP KEY HANDSHAKE

In this section we apply our key reinstallation technique against the group key handshake. We show all Wi-Fi clients are vulnerable to it, enabling an adversary to replay broadcast and multicast frames.

4.1 Details of the Group Key Handshake

Networks periodically fresh the group key, to assure that only recently authorized clients posses this key. In the most defensive case, the group key is renewed whenever a client leaves the network. The new group key is distributed using a group key handshake, and this handshake has been formally proven as secure in [32]. As shown in Figure 3, the handshake is initiated by the AP when it sends a Group Message 1 to all clients. The AP retransmits this message if it did not receive an appropriate reply. Note that the EAPOL replay counter of these retransmitted messages is always incremented by one. In our attack, the goal is to collect a retransmitted Group Message 1, block it from arriving at the client, and forward it to the client at a later point in the time. This will trick to client into reinitializing the replay counter of the installed group key.

The first prerequisite of our attack, is that clients will reinitialize the replay counter when installing an already in-use group key. Since clients also use the MLME-SETKEYS.request primitive to install the group key, this should be the case. We confirmed that in practice all Wi-Fi clients indeed reinitialize the replay counter of an already in-use group key (see Table 1 column 7). Therefore, all Wi-Fi clients are be vulnerable to our subsequent attacks.

The second prerequisite is that we must be able to collect a Group Message 1 that the client (still) accepts, and that contains a group key already in-use by the AP. Accomplishing this depends on when the AP starts using the new group key. In particular, the AP may start using the new group key immediately after sending the first Group Message 1, or it delays the installation of the group key until all clients replied using Group Message 2. Table 2, column 3, summarizes this behaviour for APs that we tested. Note that according to the standard, the new group key should be installed after all stations replied with a Group Message 2, i.e., the GTK should be installed in a delayed fashion [1, Fig. 12-53]. When the AP immediately installs the group key, our key reinstallation attack is straightforward. However, if the AP installs the group key in a delayed fashion, our attack becomes more intricate. We will discuss both these cases in more detail in Section 4.2 and 4.3, respectively.

Recall from Section 2.3 that only the AP will transmit real broadcast and multicast frames (i.e., group frames) which are encrypted using the group key. Since our key reinstallation attack targets the client, this means we cannot force nonce reusing during encryption. However, the client resets the replay counter when reinstalling the group key, which can be abused to replay frames towards clients.

Most APs refresh the group key every hour. Some networks even refresh this key whenever a client leaves the network. Additionally, clients can trigger a group key handshake by sending an EAPOL frame having the flags `Request` and `Group` [1, §12.7.7.1]. Again, Broadcom routers do not verify the authenticity of this message, meaning an attacker can forge it to trigger a group key update. All combined, we can safely assume every network will eventually execute a group key update, which we can subsequently attack.

4.2 Attacking Immediate Key Installation

Figure 7 illustrates our key reinstallation attack, when the AP immediately installs the group key after sending Group Message 1 to all clients. Notice that the group key handshakes messages themselves are encrypted using the data-confidentiality algorithm under the current PTK. On receipt of Group Message 1, the client installs the new GTK, and replies with Group Message 2. The adversary blocks this message from arriving at the AP. Hence, the AP will retransmit a new Group Message 1 in stage 2 of the attack. We now wait until a broadcast data frame is transmitted, and forward it to the victim. After this, we forward the retransmitted Group Message 1 from stage 2 to the victim. As a result, the victim will reinstall the GTK,

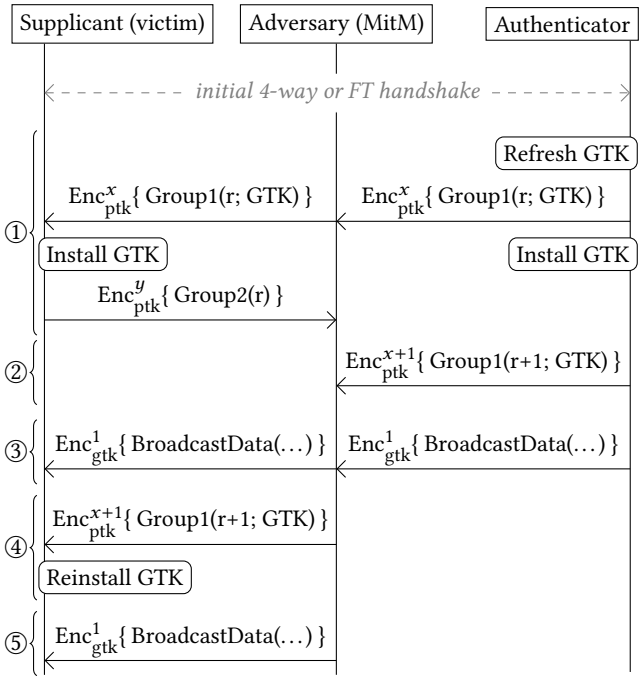

Figure 7: Key reinstallation attack against the group key handshake, when the authenticator (AP) immediately installs the GTK after sending a Group Message 1 to all clients.

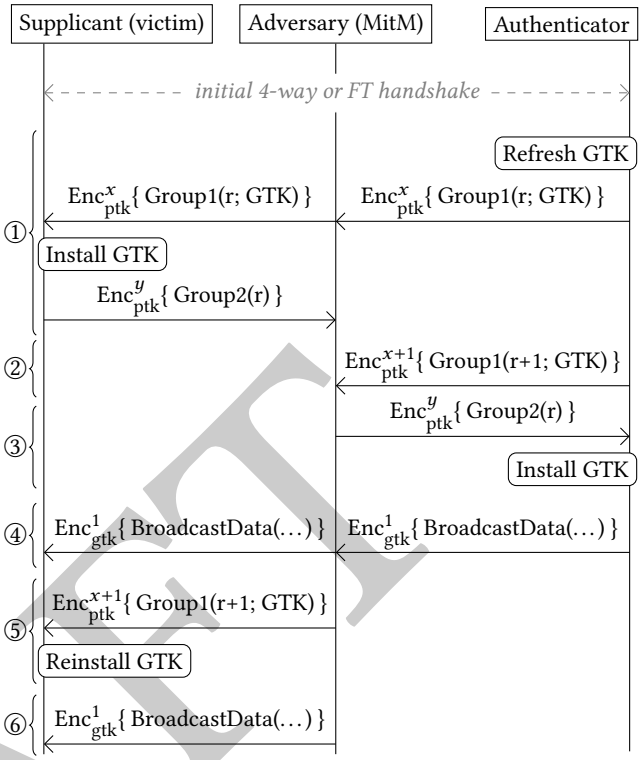

Figure 8: Key reinstallation attack against the group key handshake, when the authenticator (AP) installs the GTK after receiving a Group Message 2 from all clients.

and will thereby reinitialize its associated replay counter. This allows us to replay the broadcast data frame (see stage 5). The client accepts this frame because its replay counter was reinitialized.

It is essential that the broadcast frame we replay is sent before the retransmission of Group Message 1. This is because Group Message 1 contains the group key's current value of the replay counter (recall Section 2.5). Therefore, if it is sent after the broadcast frame, it would contain the updated replay counter and therefore cannot be abused to reinitialize the replay counter of the victim.

We confirmed this attack in practice for APs that immediately install the group key after sending Group Message 1 (see Table 2 column 3). Based on our experiments, all Wi-Fi clients are vulnerable to this attack when connected to an AP behaving in this manner.

4.3 Attacking Delayed Key Installation

Attacking the group key handshake when the AP installs the GTK in a delayed fashion is more tedious. Note that the previous attack would fail because the broadcast frame transmitted in stage 3 of Figure 7 would still be encrypted under the old group key. Indeed, at this point the AP did not yet receive Group Message 2 from the client, meaning it is still using the old group key. This is problematic because Group Message 1 (re)installs the new group key, and hence cannot be abused to reset the replay counter of the old group key.

One way to deal with this problem is illustrated in Figure 8. The first two stages of this attack are similar to the previous one. That is, the AP generates a new group key, transports it to the victim, and the adversary blocks Group Message 2 from arriving at the AP. This makes the AP retransmit Group Message 1 using an incremented EAPOL replay counter of $r + 1$. In stage 3 of the attack, however, we forward with the older Group Message 2 with replay counter value r to the AP. Interestingly, the AP should accept this message even though it does not use the latest replay counter value [1, §12.7.7.3]:

> On reception of [group] message 2, the AP verifies that the Key Replay Counter field value matches one it has used in the group key handshake.

The standard does not require that the replay counter matches the *latest* one that the AP used. Instead, it must match one that was used in the group key handshake, that is, one used in any of the (re)transmitted Group Message 1's. In practice we discovered that several implementations indeed accept this older, not yet received, replay counter (see Table 2 column 2). As a result, the AP installs the new group key. From this point on, the attack proceeds in a similar fashion as the previous one. That is, we wait until a broadcast frame is transmitted, perform the group key reinstallation in stage 5 of the attack, and then replay the broadcast frame in stage 6.

Again it is essential that the broadcast frame we want to replay is sent before the retransmission of Group Message 1. Otherwise it includes the updated replay counter of the group key.

We tested this attack against APs that install the GTK in a delayed fashion, and that accept replay counters it has used in a message to the client, but did not yet receive in a reply (recall Table 2 column 2). Note that we already know that all Wi-Fi clients reset the replay counter when reinstalling a GTK, and hence are all vulnerable.

Finally, an OpenBSD AP is not vulnerable because it installs the GTK in a delayed fashion, and only accepts the latest replay counter.

5 ATTACKING THE 802.11R FT HANDSHAKE

In this section we introduce the Fast BSS Transition (FT) handshake, and show that it is easy to break using a key reinstallation attack.

5.1 The Fast BSS Transition (FT) Handshake

Amendment 802.11r added the Fast Basic Service Set (BSS) Transition (FT) handshake to 802.11 [5]. Its goal is to reduce the roaming time when a client moves from one AP, to another one of the same encrypted network. Traditionally, this required a handshake that includes a new 802.1x and 4-way handshake (recall Figure 2). However, because the FT handshake relies on master keys derived during a previous connection with the network, a new 802.1x handshake is not required. Additionally, it embeds the 4-way handshake stage in the authentication and reassociation frames.

A normal FT handshake is shown in stage 1 of Figure 9. Observe that unlike the 4-way handshake, the FT handshake is initiated by the supplicant. The first two messages are an Authentication Request (AuthReq), and an Authentication Response (AuthResp). They are functionality equivalent to Message 1 and 2 of the 4-way handshake, respectively, and carry randomly generated nonces that will be used to derive a fresh session key. After this, the client sends a Reassociation Request (ReassoReq), and the AP replies with a Reassociaton Response (ReassoResp). They are similar in functionality to Message 3 and 4 of the 4-way handshake, finalize the FT handshake, and transport the GTK to the client.

Only the two reassociation messages are authenticated using a MIC (see Figure 9). Additionally, none of the messages in the FT handshake contain a replay counter. Instead, the FT handshake relies on the random SNonce and ANonce to provide replay protection between different invocations of the handshake [1, §13.5.2].

According to the standard, the PTK must be installed after the authentication response is sent or received [1, §13.9]. This is illustrated by the gray boxes in stage 1 of Figure 9. Additionally, the 802.1x logical port is only opened after sending or receiving the reassociation request. This assures that, even though the PTK is already installed while the handshake is still in progress, the AP and client only transmit and accept data frames once the handshake completed. However, through experiments and code inspections, we found that implementations actually install the PTK, as well as the GTK, after sending or receiving the reassociation response. This behaviour is illustrated by the black boxes in stage 1 of Figure 9.

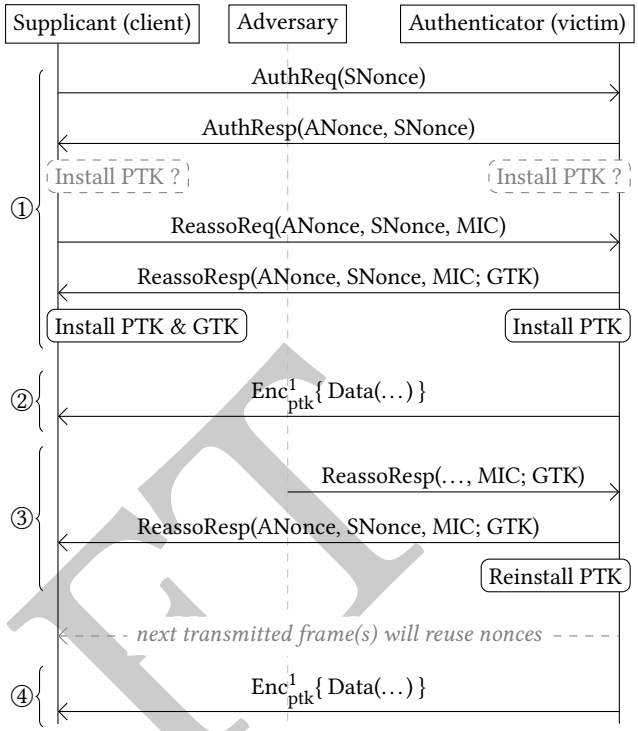

Figure 9: Key reinstallation attack against the Fast BSS Transition (FT) handshake. Note that a MitM position is not required, only the ability to eavesdrop and replay frames.

5.2 A Key Reinstallation Attack against the AP

Since the AP installs the PTK in response to a reassociation request, our goal will be to replay this frame. We remark that, according to the standard, APs must accept retransmissions of reassociation requests. This is because the reassociation response of the AP may be lost due to background noise, making the client send a new request.

Figure 9 shows the resulting key reinstallation attack against the FT handshake. Note that we do not require a man-in-the-middle position. Instead, being able to eavesdrop and inject frames is sufficient. In the first stage of the attack, we let the client and AP execute a normal FT handshake. We then wait until the AP has transmitted one or more encrypted data frames. At this point, we replay the reassociation request to the AP. Because it does not contain a replay counter, and has a valid MIC, the AP will accept and process the replayed frame. As a result, the AP will reinstall the PTK in stage 3 of the attack, thereby resetting the associated nonce and replay counter. Finally, the next data frame sent by the AP will be encrypted using an already used nonce. Similar to our previous key reinstallation attacks, this also enables an attacker to replay old data frames sent by the client to the AP. We remark that our attack is particularly devastating against the FT handshake because its messages do not contain replay counters. This enables an adversary to replay the reassociation request continuously, each time resetting both the nonce and replay counter used by the AP.

We tested this attack against all our three APs supporting 802.11r. The first is the open source `hostapd` implementation, the second is MediaTek's implementation for home routers running on a Linksys RE7000, and the third AP is a professional Ubiquiti UAP-AC-Pro. All three were vulnerable to the above key reinstallation attack.

Note that if the reassociation response is lost due to background noise, the client will retransmit the reassociation request spontaneously, causing the AP to reinstall the key. That is, without an adversary being present, APs may already be reusing nonces.

Note that messages in the FT handshake never undergo (additional) protection using a data-confidentiality protocol. In particular,

Management Frame Protection (MFP) does not protect authentication and reassociation frames [1, §12.6.19]. Hence, key reinstallation attacks against the FT handshake are trivial even if MFP is enabled.

5.3 Abusing BSS Transition Requests

An FT handshake is only performed when a station roams from one AP to another. This limits when an attack can take place. However, we can force a victim to perform an FT handshake as follows. First, assume a client is connected to an AP of a network that supports 802.11r. Then, if no other AP of this network is within range of the client, we clone a real AP of this network next to the client using a wormhole attack [33]. This makes the client think another AP of the targeted network is nearby. Finally, we wend a BSS Transition Management Request to the client. This frame is used for load balancing [1, 11.24.7], and commands the client to roam to another AP. It is an unauthenticated management frame, and hence can be forged by an adversary. Consequently, the client accepts this frame, and roams to the (wormholed) AP using an FT handshake.

We tested this against clients supporting 802.11r. This confirmed that `wpa_supplicant`, iOS [8], and Windows 10 [43] accept the transition request, and roam to another AP using an FT handshake.

6 EVALUATION AND DISCUSSION

In this section we evaluate the impact of nonce reuse for the data-confidentiality protocols of 802.11, present example attack scenarios, discuss implementation specific vulnerabilities, explain why security proofs missed our attacks, and present countermeasures.

6.1 Impact of Nonce Reuse in 802.11

The precise impact of nonce reuse caused by our attacks depends on the data-confidentiality protocol being used. Recall that this can be either TKIP, CCMP, or GCMP. All three protocol use a stream cipher to encrypt frames. Therefore, reuse of a nonce always implies reuse of the keystream. This can be used to decrypt packets. We remark that in our attack the replay counter of the victim is also reseted. Therefore, all three protocols are also vulnerable to replay attacks.

When TKIP is used, we can also recover the MIC key as follows. First, we abuse nonce reuse to decrypt a full TKIP packet, including its MIC field. Then we attack the weak Michael algorithm: given the plaintext frame and its decrypted MIC value, we can recover the MIC key [55]. Because TKIP uses a different MIC key for each communication direction (recall Section 2.4), this allows us to forge frames in one specific direction. The origin of this direction is the device targeted by the key reinstallation attack.

When CCMP is used, practical attacks are restricted to replay and decryption of packets. Although there is some work that discusses message forging attacks when nonces are repeated, the attacks are theoretic and cannot be used to forge arbitrary messages [26].

When GCMP is used, the impact is catastrophic. First, it is possible to replay and decrypt packets. Additionally, it is possible to recover the authentication key [35], which in GCMP is used to protect both communication directions (recall Section 2.4). Therefore, unlike with TKIP, an adversary can forge packets in both directions. Given that GCMP is expected to be adopted at a high rate in the next few years under the WiGig name [50], this is a worrying situation.

However, in general we can only replay, decrypt, or forge packets in a specific communication direction. The concrete direction depends on the handshake being attacked. For example, because the 4-way handshake attacks the client, it can be used to: (1) replay unicast and broadcast/multicast frames towards the client; (2) decrypt frames sent by the client to the AP; and (3) forge frames from the client to the AP. However, against the FT handshake we attack the AP instead of the client, meaning we can replay, decrypt, and/or forge packets in the reverse directions. Table 3 in the Appendix summarizes this, taking into account the handshake being attacked.

In various cases we can forge messages from the client towards the AP (see Table 3). However, the AP is generally not the final destination of a frame, and instead will forward the frame to its real destination. This means we can forge packets towards any device connected to the network. Depending on the AP, it is even possible to send a frame that is reflected back to the client.

6.2 Example Attack Scenarios

Among other things, our key reinstallation attacks allow an adversary to decrypt a TCP packet, learn the sequence number, and hijack the TCP stream to inject arbitrary data [30]. This enables one of the most common attacks over Wi-Fi networks: injecting malicious data into an unencrypted HTTP connection.

The ability to replay broadcast and multicast frames, i.e., group frames, is also a clear security violation. To illustrate how this could impact real systems, consider the Network Time Protocol (NTP) operating in broadcast mode. In this mode, the client first goes through an initialization process, and then synchronizes its clock by listening to authenticated broadcast NTP packets [44]. Malhotra and Goldberg have shown that if these broadcast frames are replayed, victims get stuck at a particular time forever [40]. Using our group key attack, we can replay these frames even if they are sent over an encrypted Wi-Fi network. Note that manipulating the time in this manner undermines the security of, for example, TLS certificates [36, 45, 52], DNSSEC [39], Kerberos authentication [39], and bitcoin [20]. These examples illustrate that the impact of replaying broadcast or multicast frames should not be underestimated.

6.3 Android 6.0 and wpa_supplicant

Our key reinstallation attack against the 4-way handshake uncovered special behavior in `wpa_supplicant`. First, version 2.3 and lower are vulnerable to all our attacks, without unexpected side-effects. However, we found that version 2.4 and 2.5 install an all-zero encryption key (TK) when receiving a retransmitted Message 3. This vulnerability seems to be caused by a remark in the 802.11 standard that appears to suggests to clear the TK 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 [42]. However, when making this patch only a benign scenario was considered where Message 3 got retransmitted because Message 4 was lost due to background interference. 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 a security sensitive, and was not backported to older versions.

Independent of this, all versions reinstall the group key when receiving a retransmitted Message 3, and are vulnerable to the group key attack of Section 4.

Because Android internally uses a slightly modified version of `wpa_supplicant`, it is also affected by our attacks. Here, Android 6.0 (used by 31.2% of Android devices [27]) and Android Wear 2.0 contain the bug causing the installation of an all-zero key.

6.4 Limitations of the Security Proofs

Interestingly, our attacks do not violate the security properties proven in formal analysis of the 4-way and group key handshake.

First, He et al. proved that the 4-way handshake provides key secrecy and session authentication [32]. Key secrecy states that only the authenticator and supplicant will posses the PTK. Since we did not recover the PTK, this properly still holds. Session authentication was proven using the standard notion of matching conversations [32]. Intuitively, this says a protocol is secure if the only way that an adversary can get a party to complete the protocol is by faithfully relaying messages [11]. Our attacks, including the channel-based MitM position we employ, do not violate this property: we can only make endpoints complete the handshake by forwarding (retransmitted) messages.

Second, He et al. proved key ordering and key secrecy for the group key handshake [32]. Key ordering assures that supplicants do not install an old GTK. This remains true in our attack, since we reinstall the *current* group key. Additionally, we do not learn the group key, hence key secrecy is also not violated by our attacks.

However, the proofs do not model key installation. Put differently, they do not state when the key is installed for use by the data-confidentiality protocol. In practice, this means the same key can be installed multiple times, thereby resetting associated nonces and/or replay counters used by the data-confidentiality protocol.

6.5 Countermeasures

Key reinstallation attacks can be mitigated at two layers. First, the entity implementing the data-confidentiality protocol can check whether an already in-use key is being installed. If so, it should not reset associated nonces and replay counters. Although this prevents our attacks, it does not defend against cases where an attacker can temporarily install a different key before reinstalling an old one.

A more robust solution is to assure that a particular key is only installed once into the entity implementing the data-confidentiality protocol during a handshake execution. For example, the generated session key in a 4-way handshake should only be installed once. When the client receives a retransmitted Message 3, it should reply, but not reinstall the session key. Note that this is precisely what version 2.6 and higher of `wpa_supplicant` is doing.

We are currently notifying vendors about the vulnerabilities we discovered, such that they can implement these countermeasures.

7 RELATED WORK

The security of the 4-way handshake has been studied in several works [31, 32, 46, 60]. This revealed a denial-of-service vulnerability [31, 46], and led to the standardization of a sightly improved design [1]. Then in 2005, He et al. presented a formal correctness proof of the 4-way handshake and of the group key handshake [32]. We showed it is essential to also model when keys are installed for use by the data-confidentiality protocol, otherwise key reinstallation attacks may be possible.

The FT handshake is based on the 4-way handshake [5]. However, there are no formal security analysis of it. Instead, existing works focus on the performance of the handshake, examples being [10, 38].

Several works investigate enterprise authentication mechanisms which negotiate master keys [16, 18, 51]. This phase generally relies on establishing a secure TLS session, meaning recent attacks on TLS are affected by it as well, examples being [9, 13, 14, 22, 53]. In this paper we studied handshakes that derive fresh session keys from such a negotiated or pre-shared master key.

The first attack on WPA-TKIP was found by Beck and Tews [55], and has been further improved in several works [29, 56–58]. Researchers also attacked the weak per-packet key construction of TKIP by exploiting biases in RC4 [6, 48]. Nowadays TKIP is deprecated by the Wi-Fi Alliance due to its security issues [61].

The GCM cipher is known to be weak when short authentication tags are used [24], and when nonces are reused [35]. Böck et al. empirically investigate nonce reuse when GCM is used in TLS [15], and discovered several servers that reuse nonces. Our attack on GCMP in 802.11 is unique because we can control when an endpoint reuses a nonce, and because GCMP uses the same key in both communication directions. Several cryptographers recently also referred to GCM as fragile [28, 47].

Finally, several researchers discovered security issues in either Wi-Fi implementations or surrounding technologies. For example, design flaws were discovered in Wi-Fi Protected Setup (WPS) [59], vulnerabilities were found in drivers [12, 17], routers were found to be using predictable pre-shared keys [37], and so on.

8 CONCLUSION

Despite the security proof of both the 4-way and group key handshake, we showed that they are vulnerable to key reinstallation attacks. These attacks do not violate the security properties of the formal proofs, but highlight limitations of the models employed by them. In particular, the models do not specify when a key should be installed for usage by the data-confidentiality protocol. Additionally, we showed that the PeerKey and fast BSS transition handshake are vulnerable to key reinstallation attacks.

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 Android 6.0 our attack triggered the installation of an all-zero key, completely voiding any security guarantees.

Rather worryingly, our key reinstallation attack even occurs spontaneously if certain handshake messages are lost due to background noise. This means that under certain conditions, implementations are reusing nonces without an adversary being present.

An interesting future research direction is to determine whether other protocol implementations are also vulnerable to key reinstallation attacks. Protocols that appear particularly vulnerable are

those that must take into account that messages may be lost. After all, these protocols are explicitly designed to process retransmitted frames, and are possibly reinstalling keys while doing so.

REFERENCES

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

[2] IEEE Std 802.11ac. 2013. *Amendment 4: Enhancements for Very High Throughput for Operation in Bands below 6 GHz.*

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

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

[5] IEEE Std 802.11r. 2008. *Amendment 2: Fast Basic Service Set (BSS) Transition.*

[6] Nadhem J AlFardan, Daniel J Bernstein, Kenneth G Paterson, Bertram Poettering, and Jacob CN Schuldt. 2013. On the Security of RC4 in TLS.. In *Usenix security*, Vol. 2013. Washington DC, USA.

[7] Wi-Fi Alliance. 2010. *Hotspot 2.0 (Release 2) Technical Specification v1.1.0.*

[8] Apple. 2017. Wi-Fi network roaming with 802.11k, 802.11r, and 802.11v on iOS. (2017). Retrieved May 19, 2017 from https://support.apple.com/en-us/HT202628

[9] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J Alex Halderman, Viktor Dukhovni, et al. 2016. DROWN: breaking TLS using SSLv2. In *25th USENIX Security Symposium (USENIX Security 16)(Aug. 2016).*

[10] Sangeetha Bangolae, Carol Bell, and Emily Qi. 2006. Performance study of fast BSS transition using IEEE 802.11 r. In *Proceedings of the 2006 international conference on Wireless communications and mobile computing*. ACM, 737–742.

[11] Mihir Bellare and Phillip Rogaway. 1993. Entity authentication and key distribution. In *Annual International Cryptology Conference*. Springer, 232–249.

[12] Gal Beniamini. 2017. Over The Air: Exploiting Broadcom's Wi-Fi Stack. (2017). Retrieved May 19, 2017 from https://googleprojectzero.blogspot.be/2017/04/over-air-exploiting-broadcoms-wi-fi_4.html

[13] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. 2015. A messy state of the union: Taming the composite state machines of TLS. In *IEEE SP*.

[14] Karthikeyan Bhargavan and Gaëtan Leurent. 2016. On the practical (in-) security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 456–467.

[15] 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*.

[16] Sebastian Brenza, Andre Pawlowski, and Christina Pöpper. 2015. A practical investigation of identity theft vulnerabilities in eduroam. In *WiSec*.

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

[18] Aldo Cassola, William Robertson, Engin Kirda, and Guevara Noubir. 2013. A Practical, Targeted, and Stealthy Attack Against WPA Enterprise Authentication. In *NDSS Symp.*

[19] Cisco. 2008. Wireless-G Exterior Access Point with Power Over Ethernet Business Series: User Guide. (2008). Retrieved May 17, 2017 from http://www.cisco.com/c/dam/en/us/td/docs/wireless/access_point/csbap/wap200e/administration/guide/WAP200E_V10_UG_C_web.pdf

[20] corbixgwelt. 2011. Timejacking & Bitcoin: The Global Time Agreement Puzzle. (2011). Retrieved May 13, 2017 from http://culubas.blogspot.be/2011/05/timejacking-bitcoin_802.html

[21] dd wrt. 2017. QCA Wireless Settings: Key Renewal Interval. (2017). Retrieved May 17, 2017 from https://www.dd-wrt.com/wiki/index.php/QCA_wireless_settings#Key_Renewal_Interval

[22] Joeri De Ruiter and Erik Poll. 2015. Protocol state fuzzing of TLS implementations. In *USENIX Security*.

[23] Morris Dworkin. 2007. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) for confidentiality and authentication. In *NIST Special Publication 800-38D*.

[24] Niels Ferguson. 2005. Authentication weaknesses in GCM. *Comments submitted to NIST Modes of Operation Process* (2005). Retrieved May 16, 2017 from http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/CWC-GCM/Ferguson2.pdf

[25] Scott Fluhrer, Itsik Mantin, and Adi Shamir. 2001. Weaknesses in the key scheduling algorithm of RC4. In *SAC*.

[26] Pierre-Alain Fouque, Gwenaëlle Martinet, Frédéric Valette, and Sébastien Zimmer. 2008. On the Security of the CCM Encryption Mode and of a Slight Variant. In *Applied Cryptography and Network Security*.

[27] Google. 2017. Dashboards: Platform Versions. (2 May 2017). Retrieved May 15, 2017 from https://developer.android.com/about/dashboards/index.html

[28] Shay Gueron and Vlad Krasnov. 2014. The fragility of aes-gcm authentication algorithm. In *11th International Conference on Information Technology: New Generations (ITNG)*. IEEE, 333–337.

[29] 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.

[30] B. Harris and R. Hunt. 1999. Review: TCP/IP security threats and attack methods. *Computer Communications* 22, 10 (1999), 885–897.

[31] Changhua He and John C Mitchell. 2004. Analysis of the 802.1 i 4-Way Handshake. In *WiSe*. ACM.

[32] 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*.

[33] Yih-Chun Hu, Adrian Perrig, and David B Johnson. 2006. Wormhole attacks in wireless networks. *IEEE journal on selected areas in communications* 24, 2 (2006), 370–380.

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

[35] 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).

[36] J. Klein. 2013. Becoming a time lord - implications of attacking time sources. In *Shmoocon Firetalks*.

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

[38] Przemysław Machań and Jozef Wozniak. 2013. On the fast BSS transition algorithms in the IEEE 802.11 r local area wireless networks. *Telecommunication Systems* 52, 4 (2013), 2713–2720.

[39] Aanchal Malhotra, Isaac E Cohen, Erik Brakke, and Sharon Goldberg. 2016. Attacking the Network Time Protocol. (2016).

[40] Aanchal Malhotra and Sharon Goldberg. 2016. Attacking NTP's Authenticated Broadcast Mode. *ACM SIGCOMM Computer Communication Review* 46, 1 (2016), 12–17.

[41] Jouni Malinen. 2015. 802.11e support? (2015). Retrieved May 17, 2017 from http://lists.shmoo.com/pipermail/hostap/2015-June/032952.html

[42] Jouni Malinen. 2015. Fix TK configuration to the driver in EAPOL-Key 3/4 retry case. Hostap commit `ad00d64e7d88`. (1 Oct. 2015).

[43] Microsoft. 2017. Fast Roaming with 802.11k, 802.11v, and 802.11r. (2017). Retrieved May 19, 2017 from https://docs.microsoft.com/en-us/windows-hardware/drivers/network/fast-roaming-with-802-11k--802-11v--and-802-11r

[44] D. Mills, J. Martin, J. Burbank, and W. Kasch. 2010. *Network Time Protocol Version 4: Protocol and Algorithms Specification.*

[45] David L Mills. 2011. *Computer network time synchronization* (2 ed.). CRC Press.

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

[47] Kenneth G. Paterson. 2015. Countering Cryptographic Subversion. (2015). Retrieved May 16, 2017 from https://hyperelliptic.org/PSC/slides/paterson-PSC.pdf

[48] Kenneth G Paterson, Bertram Poettering, and Jacob CN Schuldt. 2014. Plaintext recovery attacks against WPA/TKIP. In *International Workshop on Fast Software Encryption*. Springer, 325–349.

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

[50] Grand View Research. 2017. Wireless Gigabit (WiGig) Market Size To Reach $7.42 Billion By 2024. (2017). Retrieved May 10, 2017 from http://www.grandviewresearch.com/press-release/global-wireless-gigabit-wigig-market

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

[52] J. Selvi. 2015. Breaking SSL using time synchronisation attacks. In *DEF CON Hacking Conference*.

[53] Juraj Somorovsky. 2016. Systematic Fuzzing and Testing of TLS Libraries. In *CCS*.

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

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

[56] 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).

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

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

[59] Stefan Viehböck. 2011. Brute forcing Wi-Fi protected setup. (2011). Retrieved May 9, 2017 from http://packetstorm.foofus.com/papers/wireless/viehboeck_wps.pdf

[60] Li Wang and Balasubramaniam Srinivasan. 2010. Analysis and improvements over DoS attacks against IEEE 802.11i standard. In *NSWCTC*.

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

APPENDIX A

Table 3: Impact of our key reinstallation attack against the 4-way, FT, and group key handshake, in function of the data-confidentiality protocol used. Each cell shows in which direction frames can be replayed, decrypted, or forged.

		Replay [c]	Decrypt [a]	Forge
4-way	TKIP	AP → client	client→ AP	client→ AP [b]
	CCMP	AP → client	client→ AP	/
	GCMP	AP → client	client→ AP	client↔ AP [b]
FT	TKIP	client→ AP	AP → client	AP → client
	CCMP	client→ AP	AP → client	/
	GCMP	client→ AP	AP → client	AP ↔ client [b]
Group	*any*	AP → client [c]	/	/

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

[b] With this ability, we can use the AP as a gateway to inject packets towards *any* device connected to the network.

[c] This denotes in which direction we can replay unicast and broadcast/multicast frames. For the group key handshake, only broadcast/multicast frames can be replayed.