A Formal Analysis of IEEE 802.11's WPA2: Countering the Kracks Caused by Cracking the Counters

Cas Cremers, Benjamin Kiesl, and Niklas Medinger,
CISPA Helmholtz Center for Information Security

https://www.usenix.org/conference/usenixsecurity20/presentation/cremers

This paper is included in the Proceedings of the
29th USENIX Security Symposium.
August 12–14, 2020
978-1-939133-17-5

A Formal Analysis of IEEE 802.11's WPA2: Countering the Kracks Caused by Cracking the Counters

Cas Cremers
CISPA Helmholtz Center for Information Security

Benjamin Kiesl
CISPA Helmholtz Center for Information Security

Niklas Medinger
CISPA Helmholtz Center for Information Security

Abstract

The IEEE 802.11 WPA2 protocol is widely used across the globe to protect network connections. The protocol, which is specified on more than three-thousand pages and has received various patches over the years, is extremely complex and therefore hard to analyze. In particular, it involves various mechanisms that interact with each other in subtle ways, which offers little hope for modular reasoning. Perhaps because of this, there exists no formal or cryptographic argument that shows that the patches to the core protocol indeed prevent the corresponding attacks, such as, e.g., the notorious KRACK attacks from 2017.

In this work, we address this situation and present an extensive formal analysis of the WPA2 protocol design. Our model is the first that is detailed enough to detect the KRACK attacks; it includes mechanisms such as the four-way handshake, the group-key handshake, WNM sleep mode, the data-confidentiality protocol, and their complex interactions.

Our analysis provides the first security argument, in any formalism, that the patched WPA2 protocol meets its claimed security guarantees in the face of complex modern attacks.

1 Introduction

The vast majority of consumer internet connections take place over WiFi. In practice, this means that they use the security protocol WPA2 (short for *WiFi Protected Access 2*), which is part of the IEEE 802.11 WiFi standard. While there exists a newer WPA3 standard since late 2018, the lack of WPA3 support in existing routers and end devices means that a substantial part of all end-user internet traffic passes over WPA2 connections. For internet traffic that does not use alternative layers of protection (such as TLS), WPA2 is then the only line of defense against anyone in range of the wireless connection.

Over time, the security of the WiFi standards has been a cat-and-mouse game, with attacks and fixes following each other in sometimes rapid succession (e.g., [4, 8–10, 17, 20, 24–26, 28, 32]). Initial attacks exploited rather simple design errors, but with the advent of WPA2 and a range of patches, many protocol attacks were no longer possible. In 2005, researchers published proof sketches of the core components of the WPA2 handshake protocol [18], deeming it secure. The main attack vector that remained was a brute-force offline guessing attack, which is inherent in the protocol's design.

It came therefore as a substantial shock in 2017 when Vanhoef and Piessens showed that it was possible to break the WPA2 protocol entirely without guessing the password [29]. Their attacks exploit the combination of (i) WPA2's use of nonces (short for "numbers used once") as initialization vectors for its authenticated encryption schemes, (ii) the known fact that the reuse of initialization vectors causes severe security issues, and (iii) the observation that the reinstallation of an encryption key in WPA2 updates its associated nonce/initialization vector. The attacks force the reuse of nonces by tricking a client into reinstalling a key. Hence, they are called *key-reinstallation attacks*, or *KRACKs* in short.

While Vanhoef and Piessens proposed countermeasures in [29], they argued only informally why these countermeasures would be effective. IEEE then implemented a slightly different countermeasure. However, in 2018, Vanhoef and Piessens showed new attack variants that circumvent their previously suggested countermeasures as well as the one implemented by IEEE [30]. In addition to proposing yet another range of countermeasures, they state: *"We conclude that preventing key reinstallations is harder than expected, and believe that (formally) modeling 802.11 would help to better secure both implementations and the standard itself."* Their work led to IEEE including new countermeasures in the draft 802.11 standard.

This brings us to the present: there still exists no security analysis of the WPA2 protocol, in any methodology, that is detailed enough to capture attacks such as the KRACK attacks. Consequently, we still have no better confidence in the latest WPA2 drafts than the hope that no one has found yet another attack variant.

This may come as a surprise, given that other complex modern security protocols such as TLS 1.3, Signal, and 5G AKA have received substantial detailed analysis from the academic community using a range of techniques, e.g., [2,3,5–7,11–13, 15]. So why haven't we seen similar analyses for WPA2? We conjecture that the underlying reason is that WPA2 uses a non-standard combination of nonces and counters that are shared across several mechanisms which interact in ways that are hard to predict. In particular, this includes mechanisms that might appear irrelevant for security, but actually turn out to be a potential source of vulnerabilities (such as sleep frames, as we will see later). These design choices complicate any analysis effort, and especially contrast with TLS 1.3's relatively analysis-friendly design. Perhaps because of this, no detailed systematic analysis of WPA2 has been put forward, despite its widespread global use.

In this work, we set out to rectify this situation, and develop a detailed formal model of the WPA2 design that captures intricate attacks, including the KRACK attacks and their variants. We perform an automated analysis on our model using the *Tamarin prover* [23]. We show how our model exhibits the KRACK attacks and their variants, and evaluate the proposed countermeasures. While our work was originally motivated by those attacks, our general attacker model and detailed model of the standard capture many more subtle behaviors. Ultimately, we find that some countermeasures are sufficient to cover all attacks on our model, and hence show formally that these patches indeed prevent the earlier attacks as well as a much larger class of attacks.

Our main contributions are as follows:

- We present the first detailed security analysis of the WPA2 protocol design, including the four-way handshake, group-key handshake, WNM sleep mode, and the data-confidentiality protocols used to protect messages.

- Our formal model generalizes traditional symbolic-analysis approaches of symmetric encryption by allowing the attacker to exploit the reuse of nonces in encrypted messages, thus loosening the assumption of *perfect cryptography*. This allows us to show that if we leave out the countermeasures, our formal model exhibits the key-reinstallation attacks.

- We prove that certain countermeasures, suggested by Vanhoef and Piessens to prevent key-reinstallation attacks, are indeed sufficient to guarantee secrecy of the pairwise transient key, secrecy of the group transient keys, and authentication of the four-way handshake.

All our models, proofs, and documentation to reproduce our results are available on a dedicated website corresponding to this paper [14].

Paper Organization The rest of this paper is structured as follows: In Section 2, we discuss background required to understand the rest of the paper. In particular, we give a high-level overview of the WPA2 protocol, discuss the notorious key-reinstallation attacks, and explain Tamarin—the prover used in our formal analysis. After this, we outline our formal model of WPA2 and discuss modeling decisions in Section 3. As it is impossible to discuss our entire model on just a few pages, we focus on the most important parts. In Section 4, we present our formal analysis—this includes a discussion of the security properties we proved and details on how we proved them; Section 4 is thus particularly interesting for readers with practical experience in the symbolic analysis of protocols. In Section 5, we then present the main results of our analysis before discussing related work in Section 6 and concluding in Section 7.

2 Background

2.1 Overview of WPA2

WPA2 is a protocol used for securing communication over wireless networks. Specified in the more-than-three-thousand pages long IEEE 802.11 standard [1], it allows a client (e.g., a laptop or a smartphone) to establish cryptographic keys with an access point (e.g., a router) in order to encrypt messages exchanged over a network. The IEEE standard refers to the two protocol participants as *supplicant* (on the client side) and *authenticator* (on the access-point side); for consistency, we stick to the terms used in the standard in the rest of the paper. The two most important cryptographic keys defined by WPA2 are the so-called *pairwise transient key* (*PTK*) and the *group temporal key* (*GTK*). In typical scenarios, the pairwise transient key is used to secure a supplicant's WiFi traffic. The group temporal key is used to secure broadcast messages from an authenticator to its supplicants, e.g., for IP-multicast traffic.

To establish these keys, the supplicant and the authenticator exchange messages in a predefined manner known as the *four-way handshake*. Over the course of this four-way handshake, the supplicant and the authenticator derive their pairwise transient key, starting out from a preshared secret, which could, for instance, be the password you enter when connecting to a wireless network for the first time. This preshared secret is called the *pairwise master key* (*PMK*). As part of the handshake, the authenticator also shares its current group temporal key with the supplicant. Note here that the pairwise transient key is derived from shared inputs by both the authenticator and the supplicant whereas group temporal keys are generated by the authenticator alone.

In a nutshell, an ideal execution of the four-way handshake is as follows: The authenticator and the supplicant both generate a fresh nonce which they share with each other. Each of them then combines the two nonces with the preshared secret to derive their pairwise transient key. Once the authen-

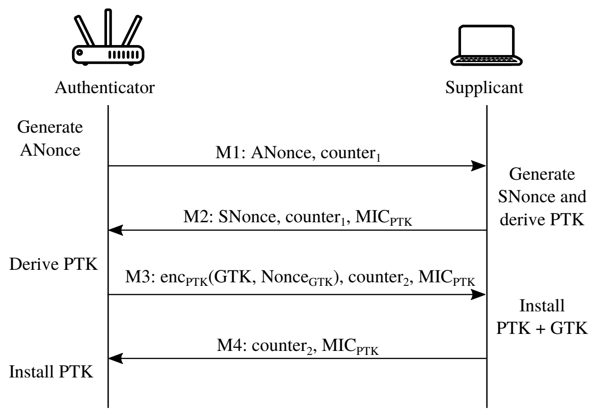

Authenticator
Supplicant
Generate ANonce
M1: ANonce, counter$_1$
Generate SNonce and derive PTK
M2: SNonce, counter$_1$, MIC$_{PTK}$
Derive PTK
M3: enc$_{PTK}$(GTK, Nonce$_{GTK}$), counter$_2$, MIC$_{PTK}$
Install PTK + GTK
M4: counter$_2$, MIC$_{PTK}$
Install PTK

Figure 1: Overview of a Successful Four-Way Handshake.

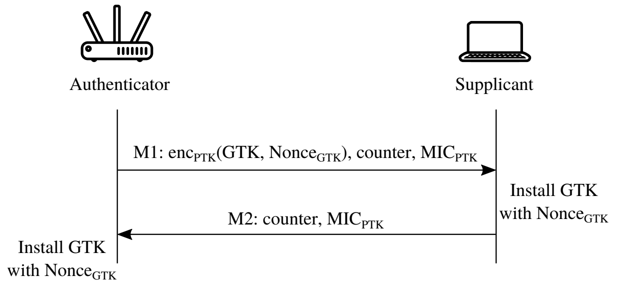

Authenticator
Supplicant
M1: enc$_{PTK}$(GTK, Nonce$_{GTK}$), counter, MIC$_{PTK}$
Install GTK with Nonce$_{GTK}$
M2: counter, MIC$_{PTK}$
Install GTK with Nonce$_{GTK}$

Figure 2: Overview of a Successful Group-Key Handshake.

ticator has derived the pairwise transient key, it shares the group temporal key with the supplicant. Note that the standard distinguishes between the *derivation* and the *installation* of a key: once a party has *derived* a key, it knows the key but it might not yet be ready to encrypt messages with that key; only when the party *installs* the key can it also encrypt messages with that key. In particular, after receiving the group temporal key, the supplicant then installs both keys and sends a confirmation to the authenticator who, upon receipt of the confirmation, also installs the pairwise transient key.

Figure 1 shows a more detailed view of an ideal four-way handshake. As shown in the figure, the handshake involves the exchange of four messages as follows:

(1) The authenticator generates a fresh nonce, called the *ANonce*, and together with a replay counter (i.e., a counter used to protect the receiver against replay attacks) sends it to the supplicant.

(2) The supplicant generates its own fresh nonce, the *SNonce*, and uses a key derivation function to derive the pairwise transient key (*PTK*) from the preshared secret (*PMK*) and the two nonces: $PTK = KDF(PMK, ANonce, SNonce)$. Then, the supplicant sends the *SNonce* and the replay counter it received in message 1 to the authenticator. Additionally, to allow the authenticator to verify the integrity of the message, it appends a *message integrity code* (*MIC*) computed with the *PTK*. In the context of the WPA2 protocol, *message integrity code* is just another term for the more common *message authentication code* (MAC).

(3) After receiving message 2, the authenticator also derives the *PTK* and checks its message integrity code. It then encrypts the *GTK* and—together with an incremented replay counter and a *MIC* (also computed with the *PTK*)—sends it to the supplicant.

(4) When the supplicant receives message 3, it checks the message integrity code. In case the check is successful, it installs the *GTK* and the *PTK*, setting the *PTK*'s *nonce* to 0; as we explain on the next page, this nonce functions as an initialization vector in the encryption scheme. To confirm to the authenticator that the installation was successful, the supplicant uses the *PTK* to compute a *MIC* for the replay counter of message 3 and sends both the replay counter and the *MIC* back to the authenticator.

At this point, the authenticator also installs the pairwise transient key and the handshake is complete. Unfortunately, this "ideal" handshake tells only part of the story. In reality, there are many more mechanisms and details that make WPA2 an immensely complex protocol with lots of room for trouble.

One mechanism, aimed at improving security, is the execution of periodic renewals of the keys, so-called *rekeys*. A rekey of the pairwise transient key involves a new iteration of the four-way handshake. A rekey of the group temporal key can involve a new iteration of the four-way handshake with one supplicant (the one who initiated the rekey) and so-called *group-key handshakes* with the other supplicants ([1], p. 2021). The purpose of such a group-key handshake is simply to distribute the new group temporal key to all supplicants.

An ideal group-key handshake is shown in Figure 2. The authenticator just sends the current group temporal key together with the current nonce and a message integrity code to the supplicant, who then installs the key and confirms the installation to the authenticator. We discuss the group-key handshake in more detail in Section 3.2.

Finally, there are other seemingly harmless mechanisms, such as the possibility to send handshake messages multiple times in order to deal with messages lost on the network, or the so-called *WNM sleep mode* (WNM is short for *wireless network management*), a mechanism that allows a supplicant to reduce power consumption by temporarily shutting itself off from certain traffic.

Ultimately, the purpose of the keys in WPA2 is to secure WiFi traffic. This is achieved by using the keys as encryption keys in so-called *data-confidentiality protocols* that protect messages exchanged over the network. These data-confidentiality protocols utilize authenticated encryption schemes based on nonces, and the wrong use (in particular the reuse) of these nonces can be exploited by attackers.

Authenticated Encryption and Nonce Reuse WPA2 allows to choose from three different data-confidentiality protocols that enable authenticated encryption ([1], p. 1953):

- TKIP (*Temporal Key Integrity Protocol*),
- CCMP (*Counter Mode with CBC-MAC Protocol*),
- GCMP (*Galois Counter Mode Protocol*).

All three use a key together with a nonce for encryption; the nonces are analogous to *initialization vectors* in counter-mode encryption: they are initialized with a certain value and then incremented for every encrypted message. On the receiver side, the nonces are also used to protect against replay attacks.

A problem that arises in this context is that the reuse of a nonce for a particular key can have negative consequences whose impact varies for the data-confidentiality protocols. However, what holds for all of them is that if a nonce is reused, then this allows an attacker to decrypt messages sent over the network as well as to replay messages. Additionally, for TKIP, nonce reuse allows an attacker to forge messages in one direction [27], and for GCMP it even allows an attacker to forge messages in *both* directions [19]. As we will see in the following, the reuse of nonces can, for instance, be caused by the reinstallation of a key.

2.2 Key-Reinstallation Attacks

In 2017, Vanhoef and Piessens demonstrated subtle attacks on WPA2 that trick a supplicant into reinstalling a key [29, 30]. Such reinstallations can seriously harm the security of WPA2 because whenever a supplicant reinstalls a key, it updates corresponding data, in particular, the nonce used for encryption.

Attacks and Countermeasures The key-reinstallation attacks by Vanhoef and Piessens are person-in-the-middle attacks that force a party into reusing a nonce by making clever use of WPA2 mechanisms such as message retransmissions. The most critical of these attacks is on the four-way handshake itself. As discussed earlier, in an ideal four-way handshake, the authenticator and the supplicant first exchange nonces before the authenticator transmits the group temporal key to the supplicant. The supplicant then installs both the pairwise transient key and the group temporal key and confirms the installation to the authenticator, who in turn also installs the pairwise transient key.

A problem arises, however, if the authenticator does not receive an installation confirmation from the supplicant, and this is where the retransmission of messages comes into play: If the authenticator does not receive an installation confirmation within a certain period of time, it assumes that the supplicant did not receive its previous message and therefore retransmits this message (M3) to the supplicant. But what if the supplicant did actually receive the previous message and

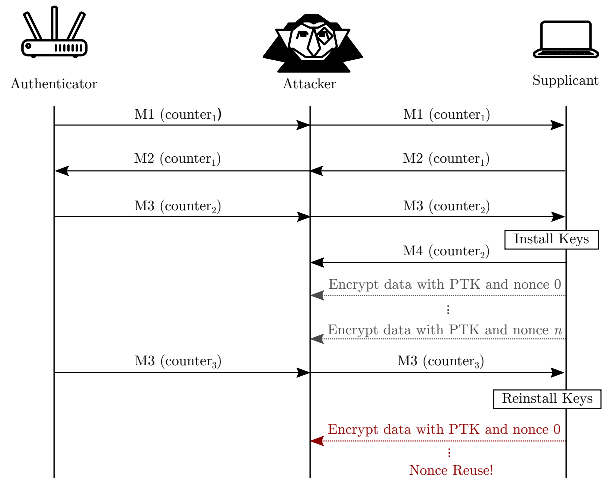


Figure 3: KRACK Attack on the Four-Way Handshake

thus installed the keys already? In that case, the supplicant would, upon receiving M3 again, reinstall the two keys and thus reset the nonce of the pairwise transient key to 0. Now, if the supplicant sent encrypted messages with the pairwise transient key before reinstalling it, the nonce reset will lead to the reuse of nonces when encrypting further messages after the second installation.

So all it takes for a person-in-the-middle attacker is to trick the authenticator into believing that the supplicant didn't install the keys. But this is easy: The attacker can simply prevent the supplicant's installation confirmation from reaching the authenticator. When the authenticator then retransmits message 3, the attacker forwards it to the supplicant, who will in turn reinstall the keys and that's it. Figure 3 illustrates an execution of this attack. In practice, the attack might not be that straightforward. This is because some implementations of the supplicant only accept encrypted messages after they installed a *PTK*, and the message (M3) the attacker intercepted is still unencrypted. Even in this case, Vanhoef and Piessens showed later [30] how to achieve a key-reinstallation by abusing the so-called *sleep-flag* of WPA2.

To prevent key-reinstallation attacks, Vanhoef and Piessens [29, 30] suggested possible countermeasures, which we discuss later in Section 5.2. Before we move on to presenting our formal model of WPA2, we give a short overview of Tamarin—the tool used for our analysis.

2.3 The Tamarin Prover

The Tamarin prover [23] is an automated-reasoning tool for the analysis of complex security protocols. Tamarin operates on the symbolic level, meaning that bit strings are abstracted to algebraic terms. Tamarin is particularly well suited for modeling complex state machines with loops and evolving state, and is therefore a natural choice for WPA2.

To formalize a security protocol in Tamarin, we encode the protocol as a collection of multiset-rewriting rules, such as:

$$\begin{array}{l}[\ \mathsf{State}(\mathit{userID},\mathit{key},\text{'READY'})\ ] \\ -[\ \mathsf{SendsReadyMsg}(\mathit{userID})\ ]\!\rightarrow \\ [\ \mathsf{Out}(\mathit{senc}(\mathit{key},\mathit{userID}))\ ]\end{array}$$

Intuitively, this rule says that if a user with a given ID and a given key is in state 'READY', it can encrypt its ID with its key and send the resulting ciphertext to the network.

Terms such as the above $\mathsf{State}(\mathit{userID},\mathit{key},\text{'READY'})$, $\mathsf{SendsReadyMsg}(\mathit{userID})$, and $\mathsf{Out}(\mathit{senc}(\mathit{key},\mathit{userID}))$ are called *facts*. Moreover, *senc* is a built-in function symbol for *symmetric encryption*. Tamarin also allows to define custom function symbols and to specify their semantics via equations, a feature we use in Section 3.4 to model the use of nonces in authenticated encryption schemes.

In general, the multiset-rewriting rules used with Tamarin consist of a *left-hand side* (the part with the fact $\mathsf{State}(\mathit{userID},\mathit{key},\text{'READY'})$ in the above example), a *right-hand side* (the part with $\mathsf{Out}(\mathit{senc}(\mathit{key},\mathit{userID}))$), and so-called *action facts* ($\mathsf{SendsReadyMsg}(\mathit{userID})$).

Once we have encoded a protocol by a set of such rules, we can specify desired properties in a guarded fragment of many-sorted first-order logic (*guarded* here means that the use of quantifiers is syntactically restricted, for details see [22]). For example, such a property could look as follows:

$$\forall\, \mathit{user}\ \mathit{key}\ t_1.\ \mathsf{Installed}(\mathit{user},\mathit{key})@t_1 \Rightarrow \neg\exists\, t_2.\ \mathsf{K}(\mathit{key})@t_2$$

This rule intuitively says that if a user has installed a certain key at time t_1, then there does not exist a time t_2 at which the attacker knows that key, or, in short: installed keys are secure.

The specification of security properties is also where the above-mentioned action facts come into play: logical formulas in Tamarin can refer to action facts and the knowledge of the attacker (denoted by the fact K as in the example) but not to facts occurring on the left-hand side or on the right-hand side of a rule. For the rule stated earlier, this means that when we write a formula, we are allowed to use the fact SendsReadyMsg but not the facts State or Out.

As underlying threat model and as a core-part of its reasoning mechanism, Tamarin assumes a Dolev-Yao attacker, i.e., a person in the middle that controls the whole network. When messages are sent to the network (with the fact Out), the attacker can learn these messages and send arbitrary messages to the nodes in the network. All this is formalized in terms of reasoning techniques in Tamarin's proof system as well as via specific rewriting rules that model the capabilities of the attacker.

Tamarin models traditionally assumed *perfect cryptography*, meaning that the attacker can only encrypt or decrypt messages (or, similarly, compute signatures, MACs, etc.) if it knows the corresponding keys. As we will discuss later (Section 3.6), we loosened the assumption of perfect cryptography in our model to allow the attacker to exploit nonce reuse in authenticated encryption schemes. Moreover, we allow the attacker to compromise certain pairwise master keys.

Once a security protocol and a security property are specified, Tamarin tries to prove the property by refuting its negation. In case Tamarin terminates, it either outputs a proof (if the statement is true) or a counter example (if the statement is false). A proof is provided in the form of a proof tree whereas a counter example is provided in the form of a trace, i.e., a sequence of steps that corresponds to a possible execution of the protocol. Proofs and counter-examples can be inspected in the graphical user interface of Tamarin.

An additional feature of Tamarin that we used heavily in our formalization of WPA2 is the possibility to specify so-called *restrictions*. A restriction is a logical formula (exactly like the formulas used to specify security properties) that must hold in every valid execution of the protocol. For example, the following formula intuitively says that a sender must increment replay counters for every message it sends:

$$\begin{array}{l}\forall\, \mathit{senderID}\ \mathit{counter}_1\ \mathit{counter}_2\ t_1\ t_2.\ (t_1 < t_2\ \wedge \\ \mathsf{SendsWithCounter}(\mathit{senderID},\mathit{counter}_1)@t_1\ \wedge \\ \mathsf{SendsWithCounter}(\mathit{senderID},\mathit{counter}_2)@t_2) \\ \ \Rightarrow \exists\, x.\ \mathit{counter}_2 = \mathit{counter}_1 + x\end{array}$$

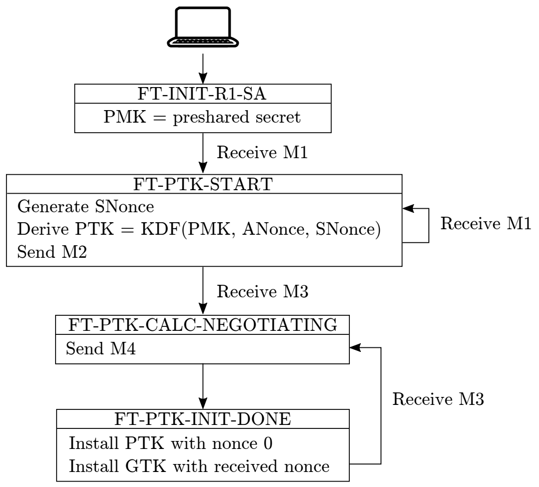
Restrictions allow to further define the semantics of a protocol in an intuitive way, often more succinctly than with only multiset-rewriting rules.

3 Formal Model of WPA2

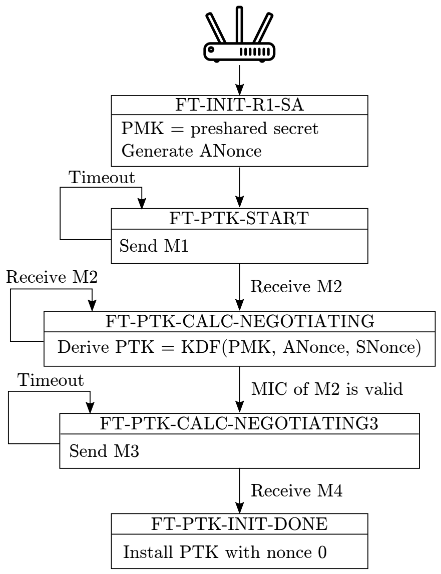
Our goal is to model the crucial components of WPA2 in a faithful way, to capture a large class of possible attacks and thus provide reliable security guarantees. In the following, we explain the core of our formal model and further details of the IEEE 802.11 standard together with notes on how we modeled them. The core mechanisms of WPA2 are specified in the standard in terms of state machines that interact with each other. In particular, the standard defines:

- two state machines for the four-way handshake (one for the supplicant and one for the authenticator),
- two state machines for the group-key handshake (again, one for the supplicant and one for the authenticator),
- one state machine that specifies how an authenticator generates new group keys.

Moreover, when a supplicant intends to enter the previously-mentioned WNM sleep mode, it has to ask the authenticator for permission. Likewise, when the supplicant wants to exit WNM sleep mode again, it has to inform the authenticator. The corresponding message exchange can be specified by two state machines, which leaves us with a total of seven state machine types, which we all capture in our formal model.



(a) Supplicant State Machine.



(b) Authenticator State Machine.

Figure 4: Simplified Four-Way Handshake State Machines

In our setting, authenticators and supplicants are modeled as devices that can start arbitrarily many concurrent threads. Thereby, a particular authenticator thread can be associated with a particular supplicant thread to establish a connection. This means that an authenticator can communicate with arbitrarily many supplicants in parallel (and vice versa), and that one and the same authenticator can start arbitrarily many sessions with one and the same supplicant. Moreover, we not only allow multiple threads per supplicant or authenticator but also multiple authenticators or supplicants as such.

To obtain strong security guarantees, we model a worst-case scenario where possible. For example, in places where the standard prescribes the use of a key that was derived from the *PTK*, we actually use the *PTK* itself. This gives the attacker more power as it can learn the full *PTK* in cases where nonce reuse would usually only allow it to learn a (less general) derived key; and any proofs we obtain give stronger guarantees for a worst-case scenario. As a beneficial side effect, this also keeps our model simpler because it contains fewer keys.

Our full formal model of WPA2 together with all proofs and extensive documentation can be downloaded from the website corresponding to this paper [14]. Due to space reasons, we do not discuss every detail of our model here. Instead, we give an overview of its critical components and how we modeled them. In particular, we focus on (1) the *four-way handshake*, (2) the *group-key handshake*, (3) *WNM sleep mode*, (4) the *encryption layer*, (5) the *replay-counter mechanisms*, and (6) our *model of nonce reuse*.

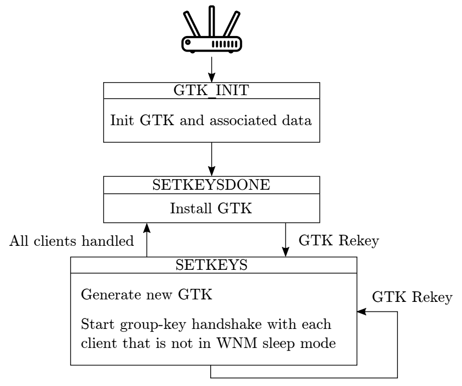
3.1 Four-Way Handshake

In Figure 4a, we show a simplified version of the supplicant state machine for the four-way handshake (defined on page 2121 of the 802.11 standard [1]). Notice that the supplicant can transition from state FT-PTK-INIT-DONE (where the keys are installed) back to the state FT-PTK-CALC-NEGOTIATING if it receives message 3. This can lead to key reinstallations.

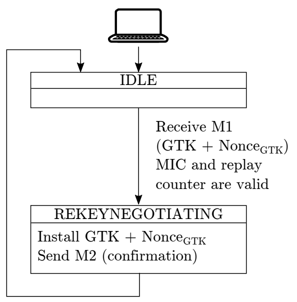
In our formal model, we encode the state machines using multiset-rewriting rules that essentially encode the transition relation between different states. For example, to encode that the supplicant transitions from state FT-PTK-START to state FT-PTK-CALC-NEGOTIATING when it receives message 3, we use the following rule (see below what the facts used in the rule stand for):

$$
\begin{array}{l}
\big[\ \mathsf{SuppState}(\sim\!\mathit{suppThreadID}, \text{'PTK\_START'}, \\
\qquad\qquad \langle \sim\!\mathit{suppID}, \sim\!\mathit{PMK}, \mathit{newPTK}, \ldots\rangle), \\
\quad \mathsf{InEnc}(\langle \mathit{m3}, \mathit{mic\_m3}\rangle, \mathit{suppThreadID}, \mathit{oldPTK}, \mathit{Supp})\ \big] \\
\ -\![\ \mathsf{SuppRcvM3}(\sim\!\mathit{suppThreadID}, \ldots), \\
\qquad \mathsf{SuppSeesCtr}(\sim\!\mathit{suppThreadID}, \sim\!\mathit{PMK}, \mathit{ctr\_m3}), \\
\qquad \mathsf{Eq}(\mathit{mic\_m3}, \mathit{MIC}(\mathit{newPTK}, \mathit{m3}))\ ]\!\!\rightarrow \\
\big[\ \mathsf{SuppState}(\sim\!\mathit{suppThreadID}, \text{'PTK\_CALC\_NEGOTIATING'}, \\
\qquad\qquad \langle \sim\!\mathit{suppID}, \sim\!\mathit{PMK}, \mathit{newPTK}, \ldots\rangle)\ \big]
\end{array}
$$

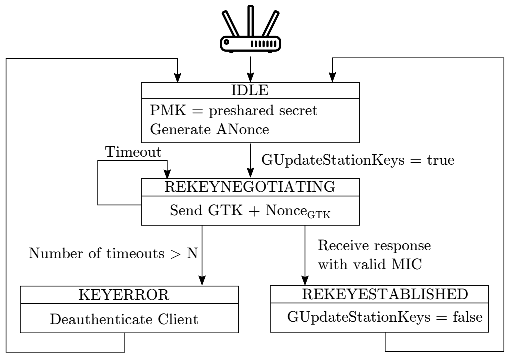
In this rule, we have a fact SuppState that represents the current state of a thread started by the supplicant. The first parameter, *suppThreadID*, uniquely identifies the supplicant and its thread. The "~" symbol is a type annotation that restricts the variable to values that were previously freshly generated (by the protocol or the attacker). The


GTK_INIT
Init GTK and associated data
SETKEYSDONE
Install GTK
All clients handled
GTK Rekey
SETKEYS
Generate new GTK
Start group-key handshake with each client that is not in WNM sleep mode
GTK Rekey

(a) Simplified Global Authenticator State Machine for GTKs.



(b) Simplified Supplicant State Machine for the Group-Key Handshake.



(c) Simplified Authenticator State Machine for the Group-Key Handshake.

Figure 5: Group-Key-Related State Machines

second parameter ('PTK_START' before the transition and 'PTK_CALC_NEGOTIATING' after the transition) is the name of the state, and the final tuple contains the (data items in the) current state, including the *PMK*, the newly derived *PTK*, and other data items.

The fact InEnc is used to receive messages from the network. Usually, in Tamarin you would model incoming messages from the network with the fact In, but in our case we have to ensure that in the initial four-way handshake, messages are not encrypted whereas in later handshakes (rekeys) they are actually encrypted. To do so, we modeled a dedicated message queue that handles the encryption mechanism. The InEnc fact is an important component of this mechanism. Later on, in Section 3.4, we explain in detail how we modeled the encryption layer.

Finally, there are the three action facts, SuppRcvM3, SuppSeesCtr, and Eq. We need the first one to prove lemmas that are required for verifying our model. The second one, SuppSeesCtr, is used to model the semantics of the replay counter mechanism via restrictions; we explain details of the replay-counter mechanism in Section 3.5. Finally, the third one, Eq, is required for making sure that the message integrity code appended to a message is valid.

Overall, we used six multiset-rewriting rules to encode the state machine of the supplicant, not including mechanisms such as key installation. The six rules correspond to the transitions in Figure 4a. The corresponding state machine for the authenticator is given in Figure 4b ([1] p. 2116).

3.2 Group-Key Handshake

As already mentioned earlier, a group-key handshake is used to distribute a group temporal key together with its nonce (and an *index*, which we do not discuss here for the sake of simplicity) to the supplicants. It involves three different state machines: two state machines (one for the supplicant and one for the authenticator) specify how messages are exchanged during a handshake whereas one other state machine on the side of the authenticator specifies how new group keys are generated and then sent to all the supplicants. We refer to this third state machine as the *global* state machine ([1] p. 2067); it is depicted in Figure 5a.

After initialization ('GTK_INIT'), the authenticator enters the state 'SETKEYSDONE'. From this state, it can transition to the state 'SETKEYS', which triggers group-key handshakes with all supplicants and thus leads to the execution of the two other state machines, depicted in Figure 5b ([1], specified implicitly on p. 2041) and Figure 5c ([1] p. 2066).

The standard specifies two ways in which the creation and distribution of a new group temporal key can be triggered:

(1) *"The Supplicant may trigger a group key handshake by sending an EAPOL-Key frame with the Request bit set to 1 and the type of the Group Key bit."* ([1] p. 2040), or

(2) *"The Authenticator may initiate the exchange when a Supplicant is disassociated or deauthenticated."* ([1] p. 2040)

We cover both cases in our model by allowing an authenticator to non-deterministically start group-key handshakes whenever it is in the 'SETKEYSDONE' state.

In our model, the state machines for the group-key handshake and the state machines for the four-way handshake can only be performed sequentially, i.e., we encode the state in a group-key handshake with the same fact symbol as the state in a four-way handshake: the fact symbol AuthState on the side of the authenticator and the state symbol SuppState on the side of the supplicant. Then, we encode transitions that lead from the 'FT-PTK-INIT-DONE' state (the state after a successful execution of the four-way handshake) to the start of a group-key handshake. The following rule shows a simplified encoding of such a transition for the authenticator, who can transition from the state ‘FT-PTK-INIT-DONE’ to the state ’REKEYNEGOTIATING’ in our model:

$$\begin{aligned}&[\ \mathsf{AuthState}(\sim\!\mathit{authThreadID}, \text{'PTK\_INIT\_DONE'}, \ldots)\ ]\\ &-\!\![\ \ ]\!\!\rightarrow\\ &[\ \mathsf{AuthState}(\sim\!\mathit{authThreadID}, \text{'REKEYNEGOTIATING'}, \ldots)\ ]\end{aligned}$$

According to the group-key state machine, the authenticator would usually enter the state ‘REKEYNEGOTIATING’ from the state ‘IDLE’. Thus, the ‘FT-PTK-INIT-DONE’ state basically takes on the role of the ‘IDLE’ state here. The consequence of this is that in our model group-key handshakes and four-way handshakes cannot be performed in parallel. We believe that this is in line with the standard, which says that, “*an Authenticator shall do a 4-way handshake before a group key handshake if both are required to be done.*” ([1] p. 2040); moreover, the replay counters used in handshake messages are specified relative to the replay counter of the first message of the respective handshake ([1], e.g., on p. 2030), which serves as another indication that group-key handshakes and four-way handshakes should not be performed in parallel.

3.3 WNM Sleep Mode

The WNM sleep mode allows a supplicant to save energy by going to sleep and thus excluding itself from group-key handshakes. If a supplicant wants to enter WNM sleep mode, it has to send a request to the authenticator. The authenticator can then, in a second message, accept the request, after which the supplicant finally goes to sleep. If the supplicant later decides it’s time to wake up again, it first sends a message to the authenticator, asking for permission to wake up. If the authenticator accepts the request, it forwards the current group key and the corresponding nonce to the supplicant. This is necessary because the supplicant didn’t participate in group-key handshakes while asleep. Figure 6 depicts the message exchange that happens when a supplicant goes to sleep and wakes up again.

In our formal model, we have dedicated state machines for the supplicant and the authenticator that allow them to perform this message exchange. In particular, we start these state

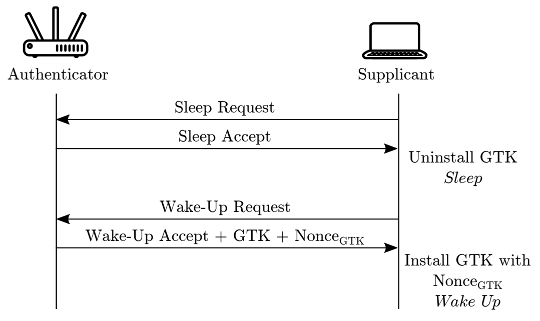


Figure 6: WNM Sleep Mode: Typical Message Exchange.

machines (both on the supplicant side and on the authenticator side) as soon as the supplicant and the authenticator have established a pairwise transient key (before, it wouldn’t make sense since WNM-related messages have to be encrypted).

To make our model as general as possible, we also decided to allow WNM-related communication to be performed in parallel to the state machines for the four-way handshake and the group-key handshake.

3.4 Encryption Layer / Message Queue

As already mentioned, messages of the first four-way handshake between an authenticator and a supplicant are not encrypted, but later handshakes—after the first installation of the pairwise transient key—are. Moreover, while the state-machines in the standard suggest that messages are sent instantly, in reality, they might be pushed into a queue and possibly sent at a later point in time.

To deal with this, we modeled a message queue that allows a sender to enqueue messages that can later be dequeued and sent to the network. Intuitively, whenever a message is dequeued, we take the currently installed pairwise transient key and encrypt the message with this key. With this queue, our model can cover cases in which a message is enqueued at some timepoint t_1 but sent—and thus encrypted with the then installed key—at a later timepoint t_2. This allows us to prove the absence of attacks caused by the sleep-flag mechanism [30].

Our basic modeling construct underlying the message queue is the so-called OutEnc fact. Usually, message transmission in Tamarin is modeled with the Out fact. For example, if we want the supplicant to send a simple message containing the string ’TEST’ over the network, we could define a multiset-rewriting rule that produces an Out fact as follows:

$$\begin{aligned}&[\ \mathsf{SuppState}(\sim\!\mathit{suppThreadID}, \ldots)\ ]\\ &-\!\![\ \ ]\!\!\rightarrow\\ &[\ \mathsf{SuppState}(\sim\!\mathit{suppThreadID}, \ldots),\\ &\quad \mathsf{Out}(\text{'TEST'})\ ]\end{aligned}$$

Here, the fact Out(’TEST’) models that the message ’TEST’ is sent to the network. With the OutEnc fact, we add an additional layer: If a sender wants to send a message, it produces an OutEnc fact that gets as parameter the ID of the sender as well as a fresh message ID. Moreover, it generates an action fact Enqueue as follows:

$$\begin{aligned}&[\ \mathsf{SuppState}(\sim\!\mathit{suppThreadID}, \ldots),\quad \mathsf{Fr}(\sim\!\mathit{messageID})\ ]\\ &-\!\![\ \mathsf{Enqueue}(\sim\!\mathit{suppThreadID}, \sim\!\mathit{messageID})\ ]\!\!\rightarrow\\ &[\ \mathsf{SuppState}(\sim\!\mathit{suppThreadID}, \ldots),\\ &\quad \mathsf{OutEnc}(\text{'TEST'}, \sim\!\mathit{suppThreadID}, \sim\!\mathit{messageID})\ ]\end{aligned}$$

The generation of the OutEnc fact does not yet denote that a message is actually sent to the network; instead, the message is only put into the message queue. A second rule then takes care of actually sending the message to the network. In addition to the OutEnc fact, this rule also takes as input the currently installed pairwise transient key to then encrypt the message with this key and a nonce and send it to the network. Note that in our model, every thread has its own independent message queue. This is a liberal interpretation of the standard, and implementations might choose a more restrictive single queue per device. Our proofs hold for both cases, because any attack on a more restrictive queue implementation would also manifest itself in our more general model.

To model the encryption with nonces, we introduced the ternary function symbol *snenc* and the binary function symbol *sndec* (in contrast to the usual binary *senc* and *sdec*). We defined the semantics of these function symbols by the equation

$$sndec(snenc(message, key, nonce), key) = message.$$

The resulting rule for sending encrypted messages then looks as follows (note that the *let/in* part is used in Tamarin to define macros)

$$\begin{array}{l}
\mathsf{let}\ \ nonce = \langle N(n), \sim sID\rangle \\
\quad newNonce = \langle N(n + \text{'1'}), \sim sID\rangle \\
\mathsf{in} \\
[\ \mathsf{OutEnc}(message, \sim sThreadID, \sim messageID) \\
\quad \mathsf{SenderPTK}(\sim ptkID, \sim sThreadID, \sim sID, PTK, nonce)\ ] \\
-[\ \mathsf{SendMessage}(\sim sThreadID, \sim messageID)\ ]\!\rightarrow \\
[\ \mathsf{Out}(snenc(message, PTK, newNonce)), \\
\quad \mathsf{SenderPTK}(\sim ptkID, \sim sThreadID, \sim sID, PTK, newNonce)\ ]
\end{array}$$

Notice the following:

(1) The rule gets the current pairwise transient key (SenderPTK) together with the current nonce. It then increments the nonce and uses it for symmetric encryption with the *PTK* and an increased nonce (*newNonce*). The result is sent to the network using a normal Out fact.

(2) The rule produces a SendMessage action fact. This fact is used together with the earlier Enqueue fact (at the place where an OutEnc fact is generated) to ensure that the queue actually follows the *first-in-first-out* principle. We achieve this by adding the following restriction to our Tamarin model:

$$\begin{array}{l}
\forall\ senderThreadID\ msgID_1\ msgID_2\ t_1\ t_2\ t_3\ t_4.\ (t_1 < t_2\ \wedge \\
\mathsf{EnqueueMessage}(senderThreadID, msgID_1)@t_1\ \wedge \\
\mathsf{EnqueueMessage}(senderThreadID, msgID_2)@t_2\ \wedge \\
\mathsf{SendMessage}(senderThreadID, msgID_1)@t_3\ \wedge \\
\mathsf{SendMessage}(senderThreadID, msgID_2)@t_4) \\
\ \Rightarrow t_3 < t_4
\end{array}$$

Intuitively, this restriction says that if a sender puts message 1 into the message queue before message 2, then message 1 has to be sent before message 2. Now the only thing that's missing is the case where a sender hasn't yet installed a pairwise transient key. This is handled by the following simple rule:

$$\begin{array}{l}
[\ \mathsf{OutEnc}(message, \sim senderThreadID, \sim msgID)\ ] \\
-[\ \mathsf{SendMessage}(\sim senderThreadID, \sim msgID)\ ]\!\rightarrow \\
[\ \mathsf{Out}(message)\ ]
\end{array}$$

Note that this rule allows the supplicant and the authenticator to send plain messages, even after the installation of a key, which could potentially lead to security violations that do not apply to the actual WPA2 protocol. However, as our analysis shows, this is not the case.

The message queue and the corresponding encryption are closely intertwined with the replay-counter mechanism, which we explain in the following.

3.5 Replay Counters

The replay counter specification in the standard can be confusing at first because there are different types of replay counters:

- The replay counters/nonces used by the authenticated encryption scheme.
- The replay counters used as core message components within handshake messages.

The replay counters used by the authenticated encryption scheme are analogous to *initialization vectors* in counter-mode encryption: They are initialized with a certain value in the beginning and then incremented for every encrypted message. Note that authenticated encryption is used both for messages encrypted with the pairwise transient key and for messages encrypted with the group temporal key.

In our model, we used the *multiset* feature of Tamarin to encode how these replay counters are incremented. A counter is seen as a multiset consisting of 1s and every increment of the counter adds another 1, like in the following rule:

$$\begin{array}{l}
[\ \mathsf{OutEnc}(message, \sim senderThreadID, \sim msgID), \\
\quad \mathsf{SenderPTK}(\sim ptkID, PTK, nonce)\ ] \\
-[\ \ ]\!\rightarrow \\
[\ \mathsf{Out}(snenc(message, PTK, nonce + \text{'1'})), \\
\quad \mathsf{SenderPTK}(\sim ptkID, PTK, nonce + \text{'1'})\ ]
\end{array}$$

On the receiver side, we model the replay-counter check with a restriction saying that whenever a message encrypted with a particular key is received, it must have a greater replay counter than any previously received message encrypted with the same key:

$$\begin{array}{l}
\forall\ keyID\ receiverID\ key\ nonce_1\ nonce_2\ t_1\ t_2.\ (t_1 < t_2\ \wedge \\
\mathsf{SeesNonce}(keyID, receiverID, key, nonce_1)@t_1\ \wedge \\
\mathsf{SeesNonce}(keyID, receiverID, key, nonce_2)@t_2) \\
\ \Rightarrow \exists\ x.\ nonce_1 + x = nonce_2\text{''}
\end{array}$$

Finally, we want to highlight that in our model, the nonces of messages sent during the four-way handshake or the group-way handshake are different from the nonces of WNM messages, which is in line with the IEEE 802.11 standard.

For the other type of replay counters, used in handshake messages (independently of encryption), the replay-counter mechanism works as follows: The authenticator appends a replay counter to a message. The supplicant is supposed to answer a particular message with the same replay counter it received. On the side of the authenticator, the standard specifies two different kinds of checks, depending on the message received:

- For message 2, the authenticator only accepts the replay counter if it equals the replay counter it used when sending message 1.
- For all other handshake messages, the authenticator accepts the replay counter if it is one of the replay counters it used in the same handshake (four-way handshake or group-key handshake).

Due to space reasons, we do not discuss here how exactly we modeled these replay counters. For details, we refer to the website corresponding to this paper [14].

3.6 Modeling Nonce Reuse

To model nonce reuse as explained in Section 2.2, we introduced a dedicated multiset-rewriting rule that allows the attacker to reveal an encryption key if it can obtain two different ciphertexts that were both encrypted with that key and with the same nonce:

$$\begin{array}{l}\mathsf{let}\ \ \mathit{encrypted\_m1} = \mathit{snenc}(\mathit{m1},\mathit{key},\mathit{nonce})\\ \qquad \mathit{encrypted\_m2} = \mathit{snenc}(\mathit{m2},\mathit{key},\mathit{nonce})\\ \mathsf{in}\\ [\ \mathsf{In}(\langle \mathit{encrypted\_m1},\mathit{encrypted\_m2}\rangle)\ ]\\ -\![\ \mathsf{Neq}(\mathit{m1},\mathit{m2}),\quad \mathsf{NonceReuse}(\mathit{key},\mathit{nonce})\ ]\!\rightarrow\\ [\ \mathsf{Out}(\mathit{key})\ ]\end{array}$$

This rule models the worst case in which *any* reuse of a nonce immediately allows the attacker to obtain the key and thus decrypt all messages sent with the same key. Note that the introduction of this rule is more general than just proving that there is no nonce reuse for a particular key: Suppose that, instead of adding this rule, we only proved that there is no nonce reuse for a particular key, then we wouldn't allow the attacker to exploit the possible nonce reuse of *other* keys to mount attacks on the protocol. By including this rule into our model, we thus make sure that our security properties are not violated by a strong attacker who can exploit nonce reuse in all possible ways.

3.7 Summary of Underlying Assumptions

The following is a summary of the assumptions made in our formal model:

(1) A single thread (of an authenticator or supplicant) cannot perform a four-way handshake and a group-key handshake in parallel.

(2) A single thread (of an authenticator or supplicant) can only start sending WNM-related messages after it has installed an initial pairwise-transient key (because WNM-related messages are encrypted with the pairwise-transient key).

(3) A single thread (of an authenticator or supplicant) can perform WNM-related communication in parallel to four-way handshakes and group-key handshakes.

(4) Every thread (of both authenticators and supplicants) has its own message queue (i.e., message queues are not *per-device* but *per-thread*).

(5) Messages that are put into a message queue are sent in the same order they were enqueued.

(6) A supplicant thread only keeps track of the latest received group key and not of multiple group keys (keeping track of multiple group keys might be required to avoid group-key reinstallations on the receiver side, which we didn't consider in our analysis; see Section 5.2 for details).

4 Analysis

After discussing the core components of our formal model, we now present details of our formal analysis. We prove the following properties for the case that countermeasures against key-reinstallation attacks are in place:

- Secrecy of the pairwise master key
- Secrecy of the pairwise transient key
- Secrecy of group temporal keys
- Authentication for the four-way handshake

We prove the latter three properties from the perspectives of both the supplicant and the authenticator. The reason for considering different perspectives is that it helps us talk about the knowledge of particular protocol participants. For instance, key secrecy from the perspective of the supplicant means something of the form, *if the supplicant has installed a key and if some other conditions hold, then the key is secret*. This means that if a supplicant knows that it installed the key and if it knows that the other conditions hold, then it can be sure that its key is secret. As the pairwise master key is not installed over the course of the protocol (but shared before), we prove its secrecy independent of any party's perspective.

Note that due to the ability of the attacker to exploit the reuse of nonces, we need to prove the absence of nonce reuse for the relevant encryption keys in our protocol. Moreover, due to the complexity of WPA2 and our corresponding model, it is impossible to prove any of the main properties directly. In fact, our whole analysis consists of around 70 lemmas (including the main properties). We provide more details about their types and intuition in Appendix A.

4.1 Secrecy of the Pairwise Master Key

Secrecy of the pairwise master key is one of the most fundamental properties within WPA2. The reason is that knowledge of a pairwise master key would allow the attacker to learn also the pairwise transient key and the group temporal keys corresponding to this pairwise master key, which would allow it to control all encrypted communication between supplicant and authenticator as well as group traffic to *all* supplicants associated with the authenticator.

To see how the attacker can learn the other two keys once it has a pairwise master key, consider the following: if the attacker observes the initial (unencrypted) four-way handshake, it can learn the corresponding *SNonce* and the *ANonce*. Thus, if it also learns the pairwise master key, *PMK*, it can derive the pairwise transient key $PTK = KDF(PMK, ANonce, SNonce)$. Once it knows the *PTK*, it can then use the *PTK* to decrypt subsequent messages, including those that contain group temporal keys.

In general, we allow the attacker to reveal pairwise master keys in order to cover cases in which certain protocol participants are compromised (e.g., because the attacker watched them type their WiFi password). Our statement for secrecy of the pairwise master key must therefore state that a pairwise master key is secret *if it has not been revealed by the attacker* (and even if other pairwise master keys have been revealed). In guarded first-order logic, we formulated this by saying that if a supplicant and an authenticator share a pairwise master key (i.e., they are associated with each other), then the attacker can only know the key if it has been revealed:

$$\begin{aligned}&\forall\, auth\ authThread\ supp\ suppThread\ PMK\ t_1\ t_2.\\ &(\mathsf{Associate}(auth, authThread, supp, suppThread, PMK)@t_1 \wedge\\ &\mathsf{K}(PMK)@t_2)\\ &\Rightarrow \exists\, t_3.\ t_3 < t_2 \wedge \mathsf{RevealPMK}(PMK)@t_3\end{aligned}$$

Compared to the other lemmas (secrecy of the other keys and authentication), proving secrecy of the pairwise master key is simpler. The intuitive reasons for this are:

- The pairwise master key is never sent over the network. Instead, it is only used as part of the input to a key derivation function for deriving pairwise transient keys.
- The pairwise master key itself is never used as an encryption key. We therefore don't need to prove the absence of nonce reuse for this key.

For the lemmas we discuss in the following, things are unfortunately more complicated.

4.2 Secrecy of the Pairwise Transient Key

We have two different statements for the secrecy of the pairwise transient key: one from the perspective of the supplicant and the other from the perspective of the authenticator. As discussed before, we prove secrecy under the assumption that the pairwise master key between the authenticator and the supplicant has not been revealed. We do, however, allow the attacker to reveal other pairwise master keys, in particular those between the same authenticator and other supplicants. Such a key revelation could, for instance, happen in practice if an attacker first gains access to the *PMK* of some supplicant S_1 (for instance, by watching a user enter their WiFi password) and then tries to use the *PMK* of S_1 to attack another supplicant S_2.

From the viewpoint of the supplicant, the corresponding lemma thus says that *if the supplicant has installed a pairwise transient key PTK that has been derived from a pairwise master key PMK, and if PMK has not been revealed, then PTK is secret*. In guarded first-order logic, the statement looks as follows:

$$\begin{aligned}&\forall\, suppThread\ supp\ PMK\ PTK\ \ldots\ t_1.\\ &(\mathsf{SuppInstalled}(suppThread, supp, PMK, PTK, \ldots)@t_1 \wedge\\ &\neg\exists\, t_2.\ \mathsf{RevealPMK}(PMK)@t_2)\\ &\Rightarrow \neg\exists\, t_3.\ \mathsf{K}(PTK)@t_3\end{aligned}$$

The corresponding statement from the authenticator's view is then analogous, replacing SuppInstalled by AuthInstalled.

To prove secrecy of the pairwise transient key, we had to prove several lemmas that guarantee the absence of nonce reuse. In particular, we proved that no key reinstallations of the pairwise transient key are possible since such key reinstallations could lead to nonce reuse, as discussed earlier in Section 2.2. Proving the absence of nonce reuse also turned out to be clearly the most complicated part about proving the secrecy of the pairwise transient key. This is interesting insofar as earlier verification attempts of WPA2 neglected nonce reuse completely.

4.3 Secrecy of Group Temporal Keys

As with the pairwise transient key, we proved the secrecy of the group temporal keys from the perspectives of both the supplicant and the authenticator. Group temporal keys are shared between a single authenticator and a group of supplicants. This means that if only one of the supplicants is compromised (i.e., the pairwise master key it shares with the authenticator is known to the attacker), the attacker will be able to control the whole group traffic between the authenticator and *all* its supplicants. Thus, when formulating the secrecy statements for the group temporal keys, we have to assume that none of the pairwise master keys are compromised.

From the perspective of the authenticator, our respective lemma says that *if an authenticator has installed a group temporal key, and if none of the pairwise master keys have been revealed, then the group temporal key is secret*, or in guarded first-order logic:

$$\begin{aligned}&\forall\ auth\ GTK\ nonce\ index\ t_1.\\ &(\mathsf{AuthInstalledGTK}(auth, \langle GTK, nonce, index\rangle)@t_1 \wedge\\ &\neg\exists\ PMK\ t_2.\ \mathsf{RevealPMK}(PMK)@t_2)\\ &\Rightarrow \neg\exists\ t_3.\ \mathsf{K}(GTK)@t_3\end{aligned}$$

The corresponding lemma from the supplicant's point of view is similar, replacing the condition that the authenticator has installed the group temporal key (AuthInstalledGTK) with the condition that the supplicant has installed it.

Note that the secrecy of the group temporal keys depends not only on the secrecy of the pairwise master key, but also on the secrecy of pairwise transient keys because group temporal keys are encrypted with the pairwise transient key when transmitted from an authenticator to its supplicants. The secrecy proofs for the group temporal keys thus rely not only on the absence of nonce reuse but also on our lemmas that show the secrecy of the pairwise transient key.

4.4 Authentication / Injective Agreement

When it comes to authentication, we prove *injective agreement*—as defined by Lowe in his hierarchy of authentication specifications [21]—for the four-way handshake. Intuitively, this means that an authenticator's executions of the four-way handshake correspond to unique executions by a supplicant (and vice versa, since we prove injective agreement in both directions). Lowe's original definition is as follows:

"*We say that a protocol guarantees to an initiator A agreement with a responder B on a set of data items ds if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the variables in ds, and each such run of A corresponds to a unique run of B.*"

To map this definition to our setting, we define the two *agents* to be the supplicant and the authenticator. As mentioned, we prove injective agreement from two perspectives: One where the authenticator is the initiator, *A*, and the supplicant is the responder, *B*, and one where the two roles are reversed. As the set of data items, *ds*, we define the set containing the pairwise master key, the pairwise transient key, the *ANonce*, and the *SNonce*.

When the authenticator is viewed as the initiator, our formulation of injective agreement states the following: Whenever an authenticator *A* finishes a four-way handshake, apparently with supplicant *S*, then the supplicant has previously finished a four-way handshake, apparently with *A*, and the supplicant and the authenticator agree on the values of the pairwise master key, the pairwise transient key, the *ANonce*, and the *SNonce*. Moreover, each run of the four-way handshake by the authenticator corresponds to a unique run of the supplicant.

We proved the second part (runs of the authenticator correspond to *unique* runs of the supplicant) in a separate statement. For our formulation of the first part, given in the following, we used facts that denote when a party completes a run (AuthCommit and SuppCommit) and when it was running a four-way handshake (AuthRunning and SuppRunning).

This is captured by the following formula:

$$\begin{aligned}&\forall\ auth\ supp\ PMK\ ANonce\ SNonce\ PTK\ t_1.\\ &(\mathsf{AuthCommit}(auth, supp, PMK, ANonce, SNonce, PTK)@t_1 \wedge\\ &\neg\exists\ t_2.\ \mathsf{RevealPMK}(PMK)@t_2)\\ &\Rightarrow (\exists\ t_3.\ t_3 < t_1 \wedge\\ &\mathsf{SuppRunning}(supp, auth, PMK, ANonce, SNonce, PTK)@t_3)\end{aligned}$$

When the roles are reversed, the statement is analogous, with *Auth* and *Supp* swapped.

Note that on the authenticator side, a *commit* happens when the authenticator receives the fourth (i.e., the final) handshake message. In this case, things are straightforward because the fourth message acts as a confirmation to the authenticator that the supplicant has finished the run of the four-way handshake. Thus, in this case we define that the supplicant was *running* the four-way handshake if it has sent the fourth message.

When the roles are reversed, we define that a *commit* of the supplicant happens when the supplicant sends the fourth message. At this point, the supplicant cannot be sure that the authenticator has finished the whole four-way handshake; all it could possibly know is that the authenticator has sent the third message. In this case, we thus define that the authenticator was *running* the protocol if it has sent the third message.

As mentioned, these statements do not yet guarantee that for every run of an initiator there is *exactly* one run of the responder. One way to prove this is to show that for a particular *SNonce*, there can be at most one execution of the four-way handshake on the supplicant side, and similarly, that for a particular *ANonce*, there can be at most one execution of the four-way handshake on the authenticator side. This is implied by our uniqueness lemmas discussed in Appendix A. We thus get injective agreement for the four-way handshake from the perspectives of both the supplicant and the authenticator.

4.5 Analysis Summary

Except for the two authentication lemmas and one helper lemma, all lemmas (including the helper lemmas) can be proved automatically by Tamarin, which takes around two hours overall on an 8-core machine with 30 GB of memory. The proofs of most lemmas take only a few seconds, with two helper lemmas (stating that the authenticator and the supplicant do not reuse nonces when encrypting messages) taking nearly all of the time. A reason for this is that in the proofs Tamarin considers all possible combinations of cases in which encrypted messages are sent. For the authentication lemmas and one helper lemma, Tamarin needs some manual guidance during proof search. We do, however, believe that fine-tuning the model (or providing custom heuristics) would help Tamarin to prove these lemmas fully automatically.

5 Results

After having presented the details of how we built our formal model of WPA2 and how we approached different aspects of the formal analysis, we now present the results of the analysis.

5.1 Behavior Covered by our Formal Model

Our formal model covers all the standard traces for

- the four-way handshake,
- the group-key handshake,
- communication for WNM sleep mode.

Moreover, our model also covers non-standard traces. For instance, by removing the patches aimed at preventing key-reinstallation attacks, we can cover traces in which these key-reinstallation attacks are executed, thus violating secrecy properties for the corresponding keys. We explain details below.

Four-Way Handshake Our model covers not only the execution of an ideal four-way handshake as depicted in Figure 1 but also all other standard behavior. For example, messages can be sent and received multiple times in cases where the IEEE standard specifies it, and rekeys of the pairwise transient key can be performed arbitrarily often. As rekeys happen after the first installation of a pairwise transient key, the standard defines that all traffic that follows is protected by a data-confidentiality protocol. Our model captures this behavior by protecting messages in rekeys accordingly, assuming a weakest possible encryption scheme in which the reuse of nonces allows the attacker to learn a key. Note that this means that our model would also cover traces in which the encryption of protocol messages leads to the reuse of nonces.

Group-Key Handshake Our model covers group-key handshakes in a very liberal way, basically allowing all traces where an authenticator generates new group keys at any possible point in time. The authenticator can transmit new group keys (and their corresponding data) to all supplicants associated with it by performing separate group-key handshakes with all of them. As the IEEE standard allows an authenticator to transmit a group key multiple times, our model also covers traces in which such retransmissions occur.

WNM Sleep Mode We cover all traces in which a supplicant enters and leaves WNM sleep mode, involving all the messages exchanged between the supplicant and the authenticator. In particular, we model the "dangerous" case in which the authenticator transmits the current group temporal key to the supplicant when the supplicant leaves WNM sleep mode.

Property	Object	Perspective: Supp.	Auth.
Secrecy	pairwise master key	(✓)	
	pairwise transient key	✓	✓
	group temporal keys	✓	✓
Authentication	four-way handshake	✓	✓

Table 1: Properties formally proven for the patched WPA2 protocol design

5.2 Patches And Their Effectiveness

Our analysis confirms that two patches/countermeasures—suggested by Vanhoef and Piessens with the aim of preventing key-reinstallation attacks—suffice to prove all the security properties (injective agreement and secrecy of keys) that are within the scope of our analysis:

(1) A supplicant should not reset or modify the nonces of a key (pairwise transient key or group temporal key) if that key is currently installed [29].

(2) A supplicant should delete the current group temporal key before entering WNM sleep mode. [30]

Especially without the first countermeasure (which we modeled with a simple action fact that checks if the new key differs from the old key when performing a key installation), secrecy of the pairwise transient key and thus also of the group temporal key cannot be guaranteed. This is because of key-reinstallation attacks that are covered by our model.

The second measure aims at preventing group-key reinstallations on the receiver (supplicant) side. As demonstrated by Vanhoef and Piessens [30], such group-key reinstallations can allow an attacker to replay group messages to the supplicant. While such group-key reinstallations don't violate any of the security properties proved in our analysis, we want to highlight that we did neither prove that they are impossible nor did we find such reinstallation attacks. We believe that proving or disproving the absence of group-key reinstallations on the supplicant side requires significant effort and is thus part of our future work.

Crucially, our analysis also does not reveal any other attacks. In other words, the KRACK attacks and their variants seem to be the only remaining attack vector on the protocol's design. Since the patches indeed prevent those attacks, we obtain stronger confidence in WPA2's design.

As with any model, there are still potential attack vectors that are outside of our analysis. Notable examples are side channels, the wider 802.11 stack design, and the decisions made for individual implementations. Given the complexity of the standard there is substantial room for misinterpretation or errors in implementation. Table 1 summarizes our results.

5.3 Kr00k Vulnerability

The so-called *Kr00k vulnerability* [16] does not indicate a vulnerability in the IEEE standard that we analyze; rather, it is related to a flaw in the implementations of some WiFi chips. In particular, a Kr00k attack exploits that—counter to the expected behavior—some (unpatched) WiFi chips still encrypt and transmit messages after a client has been disassociated. The discovery of the Kr00k vulnerability therefore doesn't invalidate any of the results of our analysis.

6 Related Work on WPA2 Verification

As stated in the introduction, the WPA2 handshake has received surprisingly little academic verification effort compared to other widely-deployed security protocols. The notable example is [18], in which the authors study the IEEE 802.11i and TLS handshakes in a version of the so-called protocol composition logic (PCL) framework.

They consider IEEE 802.11i in the scenario where TLS is used to set up a shared secret, and model simplified versions of the TLS 1.2 protocol (with four messages), the four-way handshake, and the group key handshake. They model each protocol as a straight-line protocol, therefore omitting many transitions present in the real state machines. They then show invariants for the three protocols and a composition result that these invariants are maintained by the composition. Based on this, they report a number of results, including authentication and confidentiality of the established session keys.

The KRACK attacks cannot be discovered in their approach for multiple reasons: (i) they do not explicitly consider properties of the symmetric encryption layer; (ii) they only model the group-key sequence number but none of the other counters, and state an invariant that the group-key number monotonically increases (which does not hold for standard-compliant implementations); and (iii) their straight-line models omit the complex transitions in the standard that enable counter resets. Any one of these three simplifications independently excludes the original KRACK attacks. Furthermore, since they do not model sleep frames, the later attack variants based on sleep frames are also not considered.

In contrast, our analysis models all of these aspects. As a result we can detect all these attacks as well as prove that countermeasures guarantee their absence and that of a much larger class of attacks.

7 Conclusion

In this work, we have provided the first formal security argument, in any formalism, that the patched versions of IEEE 802.11's WPA2 indeed meet their core security requirements in the face of complex attacks.

Our model includes all the interactions between a series of complex components, and it also incorporates fine-grained properties of the symmetric encryption channel, which allows us to capture attacks such as the KRACK attacks.

While our model was initially motivated by the KRACK attacks and their variants, it is not tailored specifically to those attack traces. Instead, our model systematically captures complex aspects of the WPA2 protocol, both in terms of scope (including various modes and WNM/sleep frames) and depth (modeling the nonce-reuse weakness of the underlying ciphers) in the face of a powerful attacker. Our proofs therefore show the absence of a large class of systematically defined attacks that include, but go well beyond, the KRACK attacks.

Of course, WPA2 still allows for off-line guessing attacks, but this is a fundamental property of its protocol design. Such attacks ought to be prevented by the WPA3 protocol, which follows a very different design. Initial analysis work on WPA3 has started [31] and indeed, its design seems more amenable to cryptographic analysis. We are therefore hopeful that our analysis approach can be extended to WPA3 in the near future.

References

[1] IEEE standard for information technology—telecommunications and information exchange between systems local and metropolitan area networks—specific requirements - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-2016* (*Revision of IEEE Std 802.11-2012*), pages 1–3534, Dec 2016.

[2] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol. In *EUROCRYPT (1)*, volume 11476 of *Lecture Notes in Computer Science*, pages 129–158. Springer, 2019.

[3] David A. Basin, Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, Ralf Sasse, and Vincent Stettler. A Formal Analysis of 5G Authentication. In *Proceedings of the*

2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018, pages 1383–1396. ACM, 2018.

[4] Gal Beniamini. Over The Air: Exploiting Broadcom's Wi-Fi Stack, 2017. Retrieved Feb 2020 from https://googleprojectzero.blogspot.be/2017/04/over-air-exploiting-broadcoms-wi-fi_4.html.

[5] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *IEEE Symposium on Security and Privacy*, pages 535–552. IEEE Computer Society, 2015.

[6] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *IEEE Symposium on Security and Privacy*, pages 483–502. IEEE Computer Society, 2017.

[7] Bruno Blanchet. Composition theorems for CryptoVerif and application to TLS 1.3. In *CSF*, pages 16–30. IEEE Computer Society, 2018.

[8] Nikita Borisov, Ian Goldberg, and David A. Wagner. Intercepting mobile communications: the insecurity of 802.11. In *MobiCom*, pages 180–189. ACM, 2001.

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

[10] Aldo Cassola, William K. Robertson, Engin Kirda, and Guevara Noubir. A practical, targeted, and stealthy attack against WPA enterprise authentication. In *NDSS*. The Internet Society, 2013.

[11] Katriel Cohn-Gordon, Cas J. F. Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A Formal Security Analysis of the Signal Messaging Protocol. In *EuroS&P*, pages 451–466. IEEE, 2017.

[12] Cas Cremers and Martin Dehnel-Wild. Component-Based Formal Analysis of 5G-AKA: Channel Assumptions and Session Confusion. In *NDSS*. The Internet Society, 2019.

[13] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1773–1788. ACM, 2017.

[14] Cas Cremers, Benjamin Kiesl, and Niklas Medinger. A formal analysis of IEEE 802.11's WPA2: models and proofs. https://cispa.saarland/group/cremers/tools/tamarin/WPA2/index.html.

[15] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In *ACM Conference on Computer and Communications Security*, pages 1197–1210. ACM, 2015.

[16] ESET Experimental Research and Detection Team. Kr00k, a serious vulnerability deep inside Wi-Fi encryption. https://www.eset.com/int/kr00k/. Accessed: 2010-06-08.

[17] Finn Michael Halvorsen, Olav Haugen, Martin Eian, and Stig Fr. Mjølsnes. An improved attack on TKIP. In *NordSec*, volume 5838 of *Lecture Notes in Computer Science*, pages 120–132. Springer, 2009.

[18] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *ACM Conference on Computer and Communications Security*, pages 2–15. ACM, 2005.

[19] Antoine Joux. Authentication failures in NIST version of GCM. 2006. Retrieved 01/23/2020 from https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/joux_comments.pdf.

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

[21] Gavin Lowe. A hierarchy of authentication specifications. In *10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*, pages 31–44. IEEE Computer Society, 1997.

[22] Simon Meier. *Advancing Automated Security Protocol Verification*. PhD thesis, ETH Zürich, 2013.

[23] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[24] Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt. Plaintext recovery attacks against WPA/TKIP. In *FSE*, volume 8540 of *Lecture Notes in Computer Science*, pages 325–349. Springer, 2014.

[25] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. In *NDSS*. The Internet Society, 2002.

[26] Erik Tews and Martin Beck. Practical attacks against WEP and WPA. In *WISEC*, pages 79–86. ACM, 2009.

[27] Erik Tews and Martin Beck. Practical attacks against WEP and WPA. In David A. Basin, Srdjan Capkun, and Wenke Lee, editors, *Proceedings of the Second ACM Conference on Wireless Network Security, WISEC 2009, Zurich, Switzerland, March 16-19, 2009*, pages 79–86. ACM, 2009.

[28] Mathy Vanhoef and Frank Piessens. Predicting, decrypting, and abusing WPA2/802.11 group keys. In *USENIX Security Symposium*, pages 673–688. USENIX Association, 2016.

[29] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1313–1328. ACM, 2017.

[30] Mathy Vanhoef and Frank Piessens. Release the Kraken: New KRACKs in the 802.11 standard. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 299–314. ACM, 2018.

[31] Mathy Vanhoef and Eyal Ronen. Dragonblood: A security analysis of WPA3's SAE handshake. *IACR Cryptology ePrint Archive*, 2019:383, 2019.

[32] Stefan Viehböck. Brute forcing Wi-Fi protected setup, 2011. Retrieved Feb 2020 from `https://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf`.

A General Overview and Helper Lemmas.

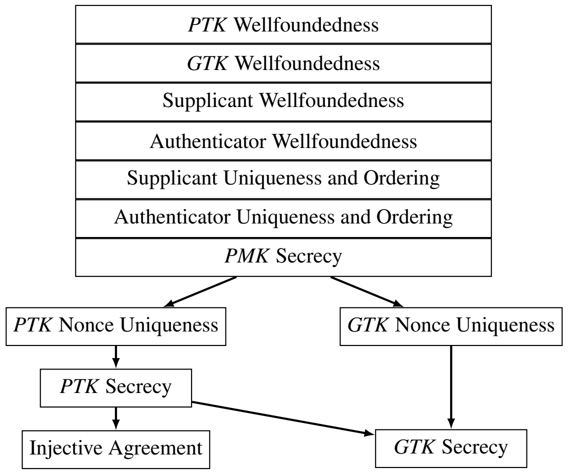


Figure 7: Lemma Map.

In Figure 7 we provide an overview of the types of lemmas that we use in our model. The core part our theory consists of lemmas which we divide into so-called *wellfoundedness lemmas*, *uniqueness lemmas*, and *ordering lemmas*. These lemmas have two main purposes:

- Characterize invariants and entry points to loops in the protocol execution.
- Help the prover with dismissing inconsistent execution traces of a protocol as early as possible to make the proof search tractable.

Wellfoundedness Lemmas The wellfoundedness lemmas are required because of the looping behavior in WPA2: the protocol specifies that nearly all messages can be sent and received multiple times in a loop. Since the Tamarin prover reasons backwards from a given assumption, we have to guide it with additional lemmas so that the backwards reasoning doesn't get stuck in a loop without ever exiting this loop again.

As a simple example, consider the following statement: *If a supplicant sent message 2, then it must have received message 1 before.* On an intuitive level, this statement is clear. However, to prove this statement, Tamarin starts with the assumption that the supplicant sent message 2, and then reasons backwards, basically asking itself "*What must have happened before the supplicant sent message 2?*" Because WPA2 allows the supplicant to send message 2 multiple times, the answer to the question involves the possibility that the supplicant just sent message 2 before. Now if Tamarin asks the same question again, the answer is again the same, and the backwards reasoning goes into a loop, because it attempts to consider all possible finite unrollings of the loop in which the supplicant repeatedly sent message 2.

The solution to this problem is to specify a lemma that basically says that there cannot be an infinite loop in which the supplicant repeatedly sends message 2, but that there must be one initial point in time at which the supplicant sent message 2 for the first time. Such a lemma can then be proved using the induction technique of Tamarin.

The situation is similar for multiple four-way handshakes: By specifying a corresponding wellfoundedness lemma, we tell Tamarin that no matter how many four-way handshakes were performed in a row, there must always be an initial four-way handshake where things have started out. Finally, we also need to specify invariants that hold at every iteration of a loop. The sum of all these statements for all the possible loops in the WPA2 model are our wellfoundedness lemmas.

Uniqueness and Ordering Lemmas When reasoning over executions of a protocol, the set of possible execution traces can quickly become gigantic, rendering the Tamarin prover practically incapable of proving statements. One reason for this is that in the most general case, the prover explores numerous traces that eventually—after spending considerable time building and analyzing these traces—turn out to be inconsistent with the semantics of the protocol.

To guide the proof search by allowing the prover to dismiss large sets of traces early on, we thus specify several *uniqueness lemmas* that guarantee that certain actions in a protocol can only happen once. Moreover, we specify *ordering lemmas* to impose order on actions. Together, these lemmas help Tamarin to focus on traces that can actually happen and to ignore the impossible ones as early on in the reasoning process as possible.