Modelling of 802.11 4-Way Handshake Attacks and Analysis of Security Properties

Rajiv Ranjan Singh[1,2][0000−0003−1808−3433], José Moreira[1][0000−0002−3210−4504], Tom Chothia[1], and Mark D. Ryan[1]

[1] School of Computer Science, University of Birmingham, Birmingham, UK
{r.r.singh, j.moreira-sanchez, t.p.chothia, m.d.ryan}@cs.bham.ac.uk
[2] Dept. of Computer Science, Shyam Lal College (Eve.), Univ. of Delhi, Delhi, India
rrsingh@shyamlale.du.ac.in

Abstract. The IEEE 802.11 standard defines a 4-way handshake between a supplicant and authenticator for secure communication. Many attacks such as KRACK, cipher downgrades, and key recovery attacks have been recently discovered against it. These attacks raise the question as to whether the implementation violates one of the required security properties or whether the security properties are insufficient. To the best of our knowledge, this is the first work that shows how to answer this question using formal methods. We model and analyse a variety of these attacks using the Tamarin prover against the security properties mandated by the standard for the 4-way handshake. This lets us see which security properties are violated. We find that our Tamarin models vulnerable to the KRACK attacks do not violate any of the standard's security properties, indicating that the properties, as specified by the standard, are insufficient. We propose an additional security property and show that it is violated by systems vulnerable to KRACK attacks, and that enforcing this property is successful in stopping them. We demonstrate how to use Tamarin to automatically test the adequacy of a set of security properties against attacks, and that the suggested mitigations make 802.11 secure against these attacks.

Keywords: IEEE 802.11 · WPA2 · 4-way handshake · Group key handshake · KRACK attack · Downgrade attack · Tamarin prover · SAPiC.

1 Introduction

The IEEE 802.11 standard [3] defines a *4-way handshake* as the key management protocol. It involves exchanging four messages between an access point (AP) and a client, or equivalently in 802.11 terminology, an authenticator and a supplicant. These exchanges enables parties to compute and share session/group keys for future unicast/multicast secure communication over the wireless medium. It also provides mutual authentication and session-key agreement.

The 4-way handshake was proven formally secure [14, 13], and had no attacks published on it until recently, when the so-called Key Reinstallation Attack

(KRACK) was uncovered by Vanhoef and Piessens in 2017 [22]. This attack exploits design and/or implementation flaws in the 4-way handshake by reinstalling already in-use session or group keys. As a consequence, the adversary can break the security guarantees, even with a secure protocol for data confidentiality, such as the AES-based Counter Cipher Mode with Block Chaining Message Authentication Code Protocol (AES-CCMP), and decrypt or replay messages [22].

Moreover, various 4-way handshake implementations have been found to be vulnerable to downgrade attacks in widely used routers [20], including models of Cisco and TP-Link. These attacks mostly affect the AP, when both the AP and the client support AES-CCMP and Temporal Key Integrity Protocol (TKIP) cipher suites. Although the client is always likely to choose the stronger AES-CCMP cipher suite over TKIP, an adversary can trick the AP into using TKIP.

We start our work by building models of 4-way handshake using the security protocol verification tool Tamarin [18]. Our modelling focuses on the subset of functionalities and messages for successful execution of the attacks on 4-way handshake, and not building a complete model of the 802.11 state machines, thus enabling a Dolev-Yao adversary [11] to exploit the vulnerabilities. We show that Tamarin can find the attacks mentioned above, and our models can formally verify that the suggested fixes to the vulnerabilities work as intended.

The IEEE 802.11 standard defines a list of security properties suggesting that it will lead to a secure 4-way handshake (e.g., freshness of session keys, secrecy of session/group keys, authentication). The existence of the attacks described above raises serious questions about these security properties: Does the IEEE 802.11 specification or some implementation violate these properties, leading to these attacks? Or are these security properties insufficient to guarantee security? If so, what security properties would be sufficient to stop the attacks? In this paper we show how these questions can be formally answered using Tamarin.

We encode the security properties from the standard using Tamarin, and use the tool to see if any of these security properties are violated in the presence of the attacks. We find that the weaknesses that lead to the KRACK attacks [22] *do not* violate any of the required properties. This suggests that the security properties, as defined in the standard, are insufficient. We then propose new security properties, and by imposing them as restrictions in Tamarin, we show that ensuring these new suggested properties is enough to stop these attacks.

We remark that our approach here is different from the normal use of formal methods for checking security protocols, which consists in defining a model of a protocol with its security properties to check for the existence of attacks. Instead, we use our models and known attacks from previous works to check if the security properties proposed in the standard are enough to ensure the security of the protocol. Where they are not, we propose a new security property that could be added to the standard, encode it in Tamarin, and use the tool to automatically show that it would be enough to stop a class of attacks, such as KRACK.

The main contributions of this work are:

– Presenting Tamarin models of the 802.11 4-way handshake that exhibit several attacks [22, 20], and formally showing correctness of suggested fixes.

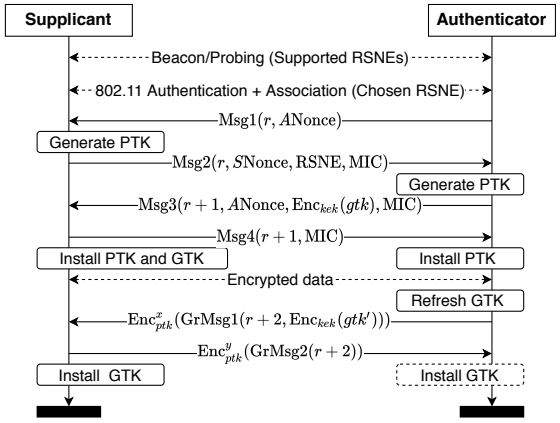


Fig. 1: IEEE 802.11 standard 4-way handshake and group key handshake

– Showing how to use Tamarin to encode the security properties defined in the standard, in order to automatically check if the weaknesses that causes any attack violate any of these properties. We show that for the KRACK attacks they do not, indicating that the current list of security properties in the standard is insufficient.
– Proposing a set of new security properties to be added to the standard, and use Tamarin to show how systems with this security property are not vulnerable to the attacks.

2 Preliminaries

The IEEE 802.11 Standard. This standard defines protocols for data confidentiality, mutual authentication, and key management, providing enhanced security at the medium access control (MAC) layer in wireless networks [3].

The original version of the standard [1] appeared in 1997, and defined the Wired Equivalent Privacy (WEP) security algorithm, based on the weak RC4 cipher. The vulnerable WEP was replaced with Wi-Fi Protected Access (WPA), as an intermediate measure, before the IEEE 802.11i amendment (WPA2) [2] was released in 2004. WPA includes the use of a message authentication code algorithm, coined as Message Integrity Check (MIC), as well as the TKIP cipher suite, which allows a more secure per-packet key system compared to the fixed key system used by WEP. The 802.11i amendment [2] and the current version of the standard [3] requires support of even more secure algorithm suites, discussed below. We summarise here the four stages of the 802.11 key generation process. We refer the reader to [3] for the full details.

– **Network Discovery.** In this stage, the clients search for available networks along with their parameters. Clients can either actively send and receive

probes, or just observe the broadcast beacons passively to learn the supported cipher suites (e.g., TKIP and/or AES-CCMP), and version of WPA. This set of parameters is called a Robust Security Network Element (RSNE).
- **Authentication and Association.** In this step, the Pairwise Master Key (PMK) is derived at both ends. In WPA2-Personal mode, the PMK is derived using a Pre-Shared Key (PSK) with a length of 8 to 63 characters, the Service Set Identifier (SSID), and the SSID length, while in WPA2-Enterprise mode, it is derived from a key generated by an Extensible Authentication Protocol (EAP), e.g., using 802.1X authentication [4]. The PMK is used later in the temporal keys generation. However, the real authentication is carried out during the 4-way handshake. The client and the AP accept or reject the association request based on the AP agreeing to the client's choice of RSNE.
- **4-Way Handshake.** The 4-way handshake takes place to agree on a fresh session key, namely the Pairwise Transient Key (PTK), and optionally the Group Temporal Key (GTK); see Fig. 1. PTK derivation [3, Sec. 12.7.1.7.5] uses the shared PMK, a supplicant nonce S_{Nonce}, an authenticator nonce A_{Nonce}, and both MAC addresses. The PTK can be refreshed after a fixed time interval, or at request from either party, by executing another 4-way handshake. The PTK is split into a Key Confirmation Key (KCK), Key Encryption Key (KEK), and Temporal Key (TK). The KCK and KEK protect handshake messages, while the TK protects data frames through the data confidentiality protocol. The 4-way handshake also transports the current GTK to the supplicant. Every message in the 4-way handshake follows the layout of EAP over LAN Key frames (EAPOL-Key) [3], and we use Msg$n$ to denote the nth message in the handshake. The authenticator starts the handshake and increments the replay counter on every message sent. The supplicant replies to messages using the received replay counter.
- **Group Key Handshake.** The standard allows for refreshing the GTK regularly, using a group key handshake, to ensure that only active clients are in possession of it. This process is initiated by the authenticator sending group message 1, denoted GrMsg1, to all clients. The clients reply, in turn, with group message 2, GrMsg2, with the received replay counter; see Fig. 1.
- **Data Confidentiality and Integrity Support.** The standard defines several data confidentiality suites such as AES-CCMP and AES-GCMP as mandatory, but also TKIP for backwards interoperability with WPA [3]. All suites include message integrity of the data frames. For brevity, we use the same notation as in [22] to denote an encrypted frame $\text{Enc}_k^n()$, being n the nonce (replay counter) in use, and k the key, i.e., PTK for unicast and GTK for broadcast messages.

We note that our focus is mainly on the attacks to the 4-way handshake. Therefore, the authentication and association stages are out of the scope of this paper, and we will hereafter assume that the PMK is already available at both ends.

Analysing Security Properties. The IEEE 802.11 standard lists five properties, labelled from a) to e), for the 4-way handshake [3, Sec. 12.6.14]. He *et*

al. [14] aggregate four out of five of these security properties into *session authentication*, which can only be asserted when *key secrecy* is guaranteed. They formalise authentication in the cryptographic model using the notion of *matching conversations* [6], guaranteeing that the two entities have consistent views of the protocol runs. Using Protocol Composition Logic (PCL) [10], they verify that such properties hold. However, PCL has been subject of criticism by some authors such as [8], as it allows one to verify authentication protocols that rely on signing, but not those relying on decryption. More disconcertingly, there are no means to establish preceding actions in a thread. In contrast to matching conversations used in [14], we use standard notions of authentication from Lowe [17], e.g., mutual, injective agreement, to verify the security properties. Moreover, in their approach using PCL [14], the authors confirm that all their proofs were constructed manually. On the other hand, our verification using Tamarin is among the first attempts to verify security properties of 802.11 automatically.

Concurrent to our work, Cremers *et al.* [9] have also developed a detailed Tamarin model of the WPA2 protocol capable of detecting KRACK attacks, among others. Though yet to appear their work, as ours, verifies the effectiveness of the patched protocol, post-discovery of the KRACK attacks, in stopping all the attacks, including the KRACK attacks. However, our goals are different; our focus is on developing a framework to test the adequacy of the required security properties in spotting the attacks. Therefore, we only model the functionalities required to demonstrate the attacks (KRACK and downgrade), rather than the whole protocol.

The Tamarin Prover and SAPiC. Tamarin is a state-of-the-art tool for symbolic verification and automated analysis of security properties in protocols, under the Dolev-Yao model [11], with respect to an unbounded number of sessions. There are similar tools for symbolic verification, most notably ProVerif [7], where protocols are specified using applied pi-calculus [5]. In our approach, we have decided to implement our models with Tamarin, since it can handle protocols with unrestricted global states and unbounded sessions. Sometimes, however, the user may have to provide auxiliary lemmas for complex protocols in order to help the tool terminate. Most importantly, Tamarin has the *restriction* feature, which allows a property to be enforced on the traces. This feature is essential for our work, to verify if enforcing particular security properties would stop an attack. To the best of our knowledge, other tools such as ProVerif do not offer this feature and hence are not suitable to our approach.

More concretely, we have developed our models using the SAPiC front-end, which allows to specify Tamarin models using processes. We provide a brief overview of these tools, but we refer the reader to [18, 16] for further reference. SAPiC parses descriptions of protocols in an extension of the applied pi-calculus [5], called *stateful applied pi-calculus*, and converts them into *(labeled) multiset rewriting rules* (MSRs) to be analysed by Tamarin.

Fig. 2 describes the SAPiC syntax. The calculus comprises an *order-sorted term algebra* with infinite sets of publicly known names PN, freshly generated

$\langle P,Q \rangle$ ::=	processes
$\mid 0$	terminal (null) process
$\mid P \mid Q$	parallel composition of P and Q
$\mid\ !P$	replication of P
$\mid\ \nu\ a;\ P$	binds a to a new fresh value in P
$\mid\ \mathsf{out}(m,t);\ P$	outputs message t on channel m
$\mid\ \mathsf{in}(m,t);\ P$	inputs of message t on channel m
$\mid$ if $Pred$ then P [else Q]	P if predicate $Pred$ holds; otherwise Q
$\mid$ event F; P	executes event (action fact) F
$\mid\ P + Q$	non-deterministic choice
$\mid$ insert m, t; P	inserts t at memory cell m
$\mid$ delete m; P	deletes the content m
$\mid$ lookup m as x in P [else Q]	if m exists, bind it to x in P; otw. Q
$\mid$ lock m; P	gain exclusive access to cell m
$\mid$ unlock m; P	waive exclusive access to m
$\mid\ [L] -[A]\!\rightarrow [R];\ P \quad (L,R,A \in \mathcal{F}^*)$	provides access to Tamarin MSRs

Fig. 2: SAPiC syntax ($a \in FN$, $x \in \mathcal{V}$, $m,t \in \mathcal{T}$, $F \in \mathcal{F}$)

names FN, and variables $\mathcal{V}$. It also comprises a signature Σ, i.e., a set of function symbols, each with an arity. The messages are elements of a set of terms $\mathcal{T}$ over PN, FN, and $\mathcal{V}$, built by applying the function symbols in Σ.

The set of facts is defined as $\mathcal{F} = \{F(t_1,\ldots,t_n) \mid t_i \in \mathcal{T}, F \in \Sigma \text{ of arity } k\}$. The special fact $K(m)$ states that the term m is known to the adversary. For a set of roles, the Tamarin MSRs define how the system, i.e., protocol, can make a transition to a new state. An MSR is a triple of the form $[L] -[A]\!\rightarrow [R]$, where L and R are the premise and conclusion of the rule, respectively, and A is a set of action facts, modelled by SAPiC events. For a process P, its trace $\mathrm{Tr}(P) = [F_1,\ldots,F_n]$ is an ordered sequence of action facts generated by firing the rules in order.

Tamarin allows to express security properties as temporal, guarded first-order formulas, modelled as trace properties. The construct $F@i$ states the presence of the fact F at time point i. A property can be specified as a *lemma* to be tested if it holds or not, and enforced as a *restriction*, while testing the other lemmas in presence of this property [18].

3 Methodology for Analysing Security Properties

We summarise our process of analysing the security properties in Fig. 3. We start by building a model of a protocol with known attacks in Sec. 4. Subsequently, we verify all the security properties listed in the standard to see if they are satisfied or violated in Sec. 5. A violated security property can then be enforced as a restriction to check if it would stop the attacks, indicating an implementation issue. Alternatively, if all the security properties are verified, but the attack still exists, we can conclude that the security properties required by the standard are insufficient and need to be augmented. After analysing the attacks, we propose

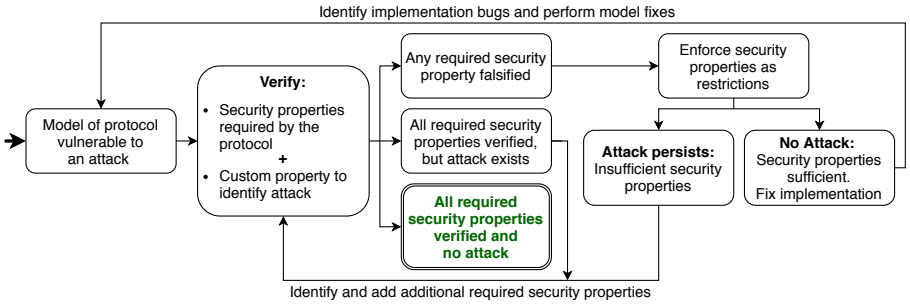


Fig. 3: Flow diagram for verifying security properties, identifying new ones, and fixing the model against an attack

a security property corresponding to the attack, shown below in Sec. 6. To test that the new property is successful in stopping the attack, we first place it as a lemma in the model and expect it to be falsified. Then, we enforce this property as a restriction in the model, expecting that it stops the attack. This helps us to verify if the attack corresponds to the new proposed security property. Finally, we execute the protocol model after fixing the vulnerability, to verify the absence of the attack. The verification of our newly proposed security properties and the fixes proves both the adequacy of the final set of properties, and correctness of the fixes in the protocol. We discuss this in Secs. 6 and 7.

4 Formal Models of the 802.11 4-Way Handshake Attacks

We present some variants of the KRACK attacks, exploiting nonce reuse [22], and a downgrade attack from [20]. Along with the attack steps, we also highlight some relevant details of our SAPiC models for the attacks and for the security lemmas corresponding to each one. Some of the details, e.g., MIC, the usage of cipher suites in encryption, or some events are omitted here due to space constraints, but they can be easily understood from the context. The complete source for the models and mechanised proofs are available at [19].

4.1 KRACK Attacks

The KRACK attacks exploit vulnerabilities in the 802.11 key management protocols [22]. An adversary tricks a victim into reinstalling an already used key by dropping, delaying or altering the order of the 4-way handshake messages between two honest principals. On every key installation, the standard mandates that the replay counter (nonce) of the data confidentiality protocol be reset. The adversary can collect different encrypted messages using the same key and nonce: messages sent after the initial key installation, and messages sent after the key reinstallation. The adversary can then use this information to attack the data confidentiality protocol. The practical implications of the attack may enable the

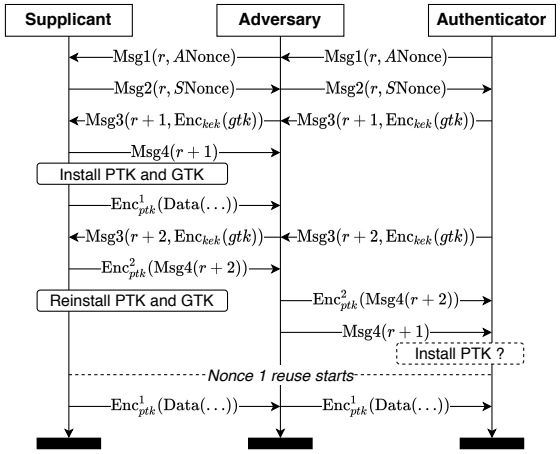


Fig. 4: KRACK - plaintext retransmission of message 3 after PTK install

adversary to replay, decrypt or even forge the data packets, depending on the choice of the cipher suite (e.g., TKIP, AES-CCMP,AES-GCMP). We refer the reader to [22, 15] for the detailed consequences of the attack.

The underlying causes of the attacks are the unclear standard specifications, such as the authenticator accepting any replay counter previously used in the 4-way handshake, not only the latest one [3, Sec. 12.7.6.5]. However, in practice, many APs fail to validate it, and imprudently accept an older replay counter.

We have successfully modelled several KRACK attacks exploiting the retransmission of message 3 and forcing nonce reuse [22]. We remark that the goal of our models is not to verify the compromise of the data confidentiality protocol. Instead, we aim at detecting the sufficient conditions that allow an adversary to exploit it, i.e., reinstallations of the same key.

Retransmission of Message 3 after PTK Install. This variant of KRACK [22, Sec. 3.3] occurs when the supplicant accepts plaintext retransmission of message 3, even after a PTK has been installed. The message flow of the attack is shown in Fig. 4, and the outline of our model of the supplicant and authenticator are in Fig. 5. Note that we prepend 'S_' and 'A_' to the events executed at the supplicant and authenticator, respectively. The main process is defined as $\nu\ pmk$; (!Supplicant | Authenticator), instantiating an arbitrary number of supplicant processes. Our model computes the PTK [3, Sec. 12.7.1.7.5] with the identifiers A_{id}, S_{id} acting as the MAC addresses as follows:

$$ptk = \text{CalcPtk}(pmk, A_{\text{Nonce}}, S_{\text{Nonce}}, A_{\text{id}}, S_{\text{id}}).$$

The adversary sits between the supplicant and the authenticator to perform a man-in-the-middle (MitM) attack, and forwards messages 1-3 normally. The event S_InstallsPtk(S_{id}, ptk) captures an initial PTK install, after which the

```
Supplicant :=
ν S_id; out(S_id);
!( in(A_id);
   in(Msg1(r, A_Nonce));
   ν S_Nonce;
   let ptk = CalcPtk(pmk, A_Nonce, ...) in
   out(Msg2(r, S_Nonce));
   in(Msg3(r + 1, A_Nonce, Enc_kek(gtk)));
   event Running(S_id, A_id, pars);
   out(Msg4(r + 1));
   event S_InstallsPtk(S_id, ptk);
   event S_InstallsGtk(S_id, gtk);
   (( event Commit(S_id, A_id, pars)
   ) +
   (  in(Msg3(r + 2, A_Nonce, Enc_kek(gtk)));
      event Running(S_id, A_id, pars);
      out(Enc_ptk(Msg4(r + 2)));
      event S_InstallsPtk(S_id, ptk);
      event S_InstallsGtk(S_id, gtk);
      event Commit(S_id, A_id, pars)
   )))
```

```
Authenticator :=
ν A_id; out(A_id);
!( in(S_id);
   ν r;
   ν A_Nonce;
   out(Msg1(r, A_Nonce));
   let ptk = CalcPtk(pmk, ...) in
   in(Msg2(r, S_Nonce));
   ν gtk;
   event Running(A_id, S_id, pars);
   event A_InstallsGtk(gtk);
   out(Msg3(r + 1, A_Nonce, Enc_kek(gtk)));
   (( in(Msg4(r + 1));
      event A_InstallsPtk(ptk);
      event Commit(A_id, S_id, pars)
   ) +
   (  out(Msg3(r + 2, A_Nonce, ...));
      in(Enc_ptk(Msg4(r + 2)));
      event A_InstallsPtk(ptk);
      event Commit(A_id, S_id, pars)
   )))
```

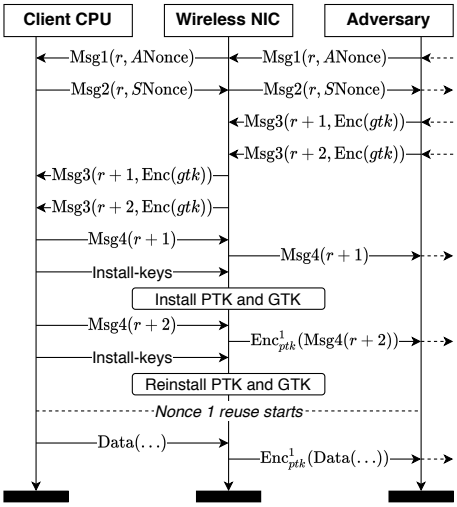
Fig. 5: Model outline for supplicant and authenticator vulnerable to KRACK attack based on plaintext retransmission of message 3

supplicant can send encrypted frames using the encryption key TK associated to PTK. Message 4 is blocked from reaching the authenticator by the adversary. The model uses the non-deterministic choice in the authenticator process via the + operator from the SAPiC calculus. Therefore, it captures either the reception of message 4, and installs the PTK, or timeouts and retransmits message 3 with an updated replay counter, and waits again for the confirmation.

Similarly, in order to capture the fact that the state machine of the supplicant accepts plaintext retransmission of message 3, we also branch the supplicant process, in order to capture traces completing a normal run of the protocol, and traces with an adversary blocking message 4. This latter case matches the attack scenario with the supplicant reinstalling an already in-use PTK (and GTK). It follows that the next data frames sent by the supplicant will be encrypted with a reused nonce. Our model, therefore, is aimed at capturing the traces with key reinstallations on the supplicant side using the same PTK already installed.

Retransmission of Message 3 before PTK Install. This KRACK attack has two variants with the supplicant accepting either a plaintext or encrypted retransmission of message 3 with the PTK yet to be installed [22, Sec. 3.4].

The first case is shown in Fig. 6. This attack assumes that the authenticator performs its actions as expected. The first two messages are transmitted normally. However, the original message 3 is blocked by the adversary while he waits for retransmitted of message 3. Both messages are then forwarded to the



(a) Message sequence diagram

Supplicant =
!(
  // *Messages 1-3 processed normally*
  ((
  // *No other msg. recv. in between*
    out(Msg4($r+1$));
    event S_InstallsPtk(S_{id}, ptk)
) + (
  // *Retransm. message 3 recv.*
    in(Msg3($r+2, A_{\mathrm{Nonce}}, \mathrm{Enc}(gtk)$));
    out(Msg4($r+1$));
    event S_InstallsPtk(S_{id}, ptk);
    out(Enc_{ptk}(Msg4($r+2$)));
    event S_InstallsPtk(S_{id}, ptk)
  )))

(b) Model outline

Fig. 6: KRACK - plaintext retransmission of message 3 before PTK install

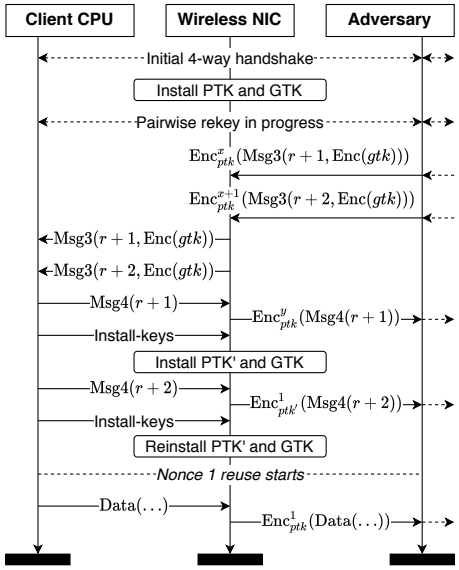
supplicant. This triggers a race condition between the CPU and the network interface controller (NIC), which causes that the same key be reinstalled. In our model for this attack, Fig. 6b, the supplicant comprises both the NIC and the CPU, and it considers two branches in order to capture an implementation vulnerable to the attack: one where the 4-way handshake follows the normal course, and another where the attacker is able to cause key reinstallation.

The second case of this attack is presented in Fig. 7. The main difference is that it can only be executed during the PTK rekey phase. After an initial successful handshake, both principals install a PTK. During the PTK rekey process, the adversary follows the same strategy as above: it waits for a retransmission of message 3. This time, the messages are encrypted under the installed PTK, but the adversary is able to identify what particular message is being sent (e.g., by timeouts or message lengths). By appropriately delaying and forwarding the messages, the adversary causes a reinstall of the PTK being refreshed, ptk'. Our model (Fig. 7b) captures an arbitrary number of PTK rekey negotiations, and, again, it branches non-deterministically to capture the transitions of a supplicant state machine vulnerable to the attack.

For all three cases above (Figs. 4, 6 and 7), we query for the absence of KRACK attacks with lemma: "given an installation of PTK by the supplicant, it is not the case that there exists an earlier installation with the same PTK,"

$$\forall id, ptk, t_1.\ \mathrm{S\_InstallsPtk}(id, ptk)@t_1 \Rightarrow \neg(\exists t_2.\ \mathrm{S\_InstallsPtk}(id, ptk)@t_2 \wedge (t_2 < t_1)). \qquad \text{(NoKrackPtk)}$$

The events S_InstallsPtk are placed in the parts of the model where the primitive MLME-SETKEYS.request [3] is called, which causes nonce reset.



(a) Message sequence diagram

Supplicant =
!(// *Initial 4-way handshake*
  event S_InstallsPtk(S_{id}, ptk);
  !(// *PTK rekey*
    // *Messages 1-3 processed normally*
    ((// *No other msg. received*
      out($\mathrm{Enc}_{ptk}(\mathrm{Msg4}(r+1))$);
      event S_InstallsPtk(S_{id}, ptk')
) + (// *Retransmitted msg. 3 recv.*
      in($\mathrm{Enc}_{ptk}(\mathrm{Msg3}(r+2,\ldots))$);
      out($\mathrm{Enc}_{ptk}(\mathrm{Msg4}(r+1))$);
      event S_InstallsPtk(S_{id}, ptk');
      out($\mathrm{Enc}_{ptk'}(\mathrm{Msg4}(r+2))$);
      event S_InstallsPtk(S_{id}, ptk')
    ))))

(b) Model outline

Fig. 7: KRACK - encrypted retransmission of message 3 before PTK install

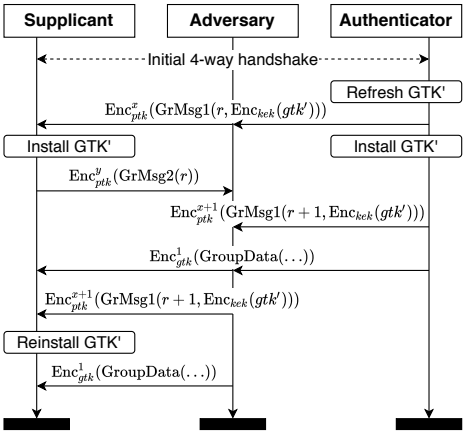
As expected, our Tamarin models [19] falsify Lemma (NoKrackPtk), proving the existence of KRACK, allowing an adversary to cause key reinstall, nonce reuse and break the security guarantees of the data confidentiality protocol.

Attack Against the Group Key Handshake. This variant of the KRACK attack targets the group key handshake, and tricks the supplicant into reinstalling a GTK, rather than a PTK [22, Sec. 4.1]. The attack is shown in Fig. 8. Note that the group key handshake runs encrypted by the already installed PTK. The standard requires that the supplicant install the GTK upon receipt of group message 1, regardless of whether it is a retransmission or not, and reply with group message 2. The adversary delays group message 2 from reaching the authenticator, triggering retransmission of group message 1. Now, the adversary forwards both versions of group message 1 to the supplicant, which causes a GTK install and subsequent reinstall. This will allow the attacker to replay group data frames to the supplicant [22].

To capture the reinstall of the GTK, Tamarin falsifies the following lemma stating that "given an installation of GTK by the supplicant, it is not the case that there exists an earlier installation with the same GTK,"

$$\forall id, gtk, t_1.\ \mathrm{S\_InstallsGtk}(id, gtk)@t_1 \Rightarrow \neg(\exists t_2.\ \mathrm{S\_InstallsGtk}(id, gtk)@t_2 \wedge (t_2 < t_1)). \qquad \text{(NoKrackGtk)}$$

Our model (Fig. 8b) captures a scenario with a supplicant accepting arbitrary number of executions of the group key handshake, as long as the group message 1



(a) Message sequence diagram

```
Supplicant =
!(
  // Initial 4-way handshake
  !(
    // Group key handshake
    in(Enc_ptk(GrMsg1(r',
                      Enc_kek(gtk'))));
    // If r' already used, finish thread
    event S_InstallsGtk(S_id, gtk');
    out(Enc_ptk(GrMsg2(r')))
  ))
```

(b) Model outline

Fig. 8: KRACK against group key handshake

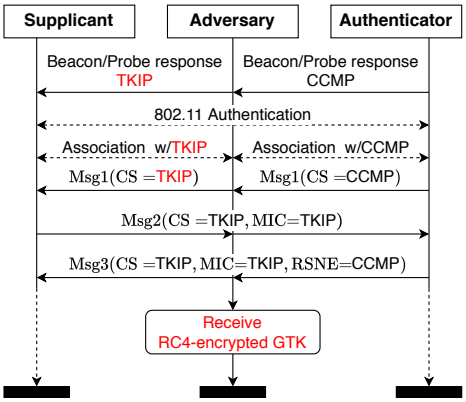
has an increased replay counter. We note that for this model we assume an initial valid 4-way handshake without exhibiting PTK reinstall.

4.2 Cipher Suite Downgrade

The downgrade attack we consider [20] is limited to the authenticator-side only. In a correct implementation, a client should be able to detect this attack easily by observing inconsistencies in the RSNE information. Recall from Sec. 2 that the RSNE information is selected in the association stage in plaintext, and subsequently encrypted and transmitted as part of message 3, as shown in Fig. 1. The supplicant must verify that the RSNE information observed in the association stage matches with the authenticated contents of message 3, and it should terminate the handshake otherwise.

In a downgrade attack, depicted in Fig. 9, the adversary forces GTK encryption with a weak cipher suite (RC4), rather than the intended strong cipher suite (AES-CCMP). The attack was discovered on the access point TP-Link WP841P [20, Sec. 5.2]. The authenticator advertises support for AES-CCMP during the association stage. However, it will follow the supplicant in switching the cipher suite in mid-handshake process, accepting the TKIP-based message 2.

An adversary acts as a MitM by negotiating the AES-CCMP suite with the authenticator, and TKIP with the supplicant, as message 1 is in plain. The supplicant calculates the PTK and replies with message 2 using the TKIP suite. The authenticator accepts the message, overrides its initial AES-CCMP selection, and responds with a well-formed TKIP message 3 containing the GTK encrypted with RC4. The adversary can now exploit the weakness of this cipher to recover the GTK [21]. The RSNE mismatch can be easily detected on forwarding of message 3 to the supplicant, which can drop the connection. Unfortunately, by this time, the adversary is already in possession of the RC4-encrypted GTK.



(a) Message sequence diagram

Authenticator =
!(ν *tid*; *// Thread ID*
in(⟨'AuthReq', S_{id}⟩);
out(⟨'AuthResp', S_{id}, 'CCMP'⟩);
in(⟨'AssocReq', S_{id}, 'CCMP'⟩);
out(⟨'AssocResp', S_{id}, 'CCMP'⟩);
// 4-way handshake
event A_Starts(*tid*, 'CCMP');
out(Msg1(r, A_{Nonce}, 'CCMP'));
in(Msg2(r, S_{Nonce}, cs));
out(Msg3($r+1$, A_{Nonce}, $\text{Enc}_{\langle cs,kek\rangle}(gtk)$));
event A_SentMsg3(*tid*, cs);
// (...)
)

(b) Model outline

Fig. 9: Downgrade Attack on 802.11 (TP-Link WP841P)

Encryption with different cipher suites can be modelled, e.g., with a signature Enc', Dec' indicating the cipher suite cs as an additional parameter. Then,

$$\forall m, k, cs.\ \text{Dec}'_k(\text{Enc}'_k(m, cs), cs) = m.$$

Note, that this theory is semantically equivalent to the usual symmetric encryption using as key the tuple $k' = \langle cs, k\rangle$, because $\text{Dec}_{\langle cs,k\rangle}(\text{Enc}_{\langle cs,k\rangle}(m)) = m$.

Our Tamarin model queries that "for each run of the protocol, the cipher suites used by them is be the same," implying that a change of the cipher suite in between a run is impossible. As expected, the lemma below is falsified:

$$\forall tid, cs_1, cs_2, t_1, t_2.\ \text{A\_SentMsg3}(tid, cs_1)@t_1 \wedge \text{A\_Starts}(tid, cs_2)@t_2 \Rightarrow (cs_1 = cs_2). \qquad \text{(NoDowngrade)}$$

5 Analysis of IEEE 802.11 Security Properties

In this section, we list the five properties a)-e) specified for the 4-way handshake in the 802.11 standard [3, Sec. 12.6.14]. These properties overlap with each other and cannot be easily encoded into conventional queries, e.g., secrecy or authentication. Therefore, we sometimes define multiple security lemmas that jointly satisfy a given property. Moreover, the standard is unclear about what properties are satisfied by the group key handshake. In that case, we consider an extension of property c) below for GTK. We recall that we prepend 'S_' and 'A_' to the supplicant and authenticator events, respectively.

a) **Confirm the existence of the PMK at the peer.** As stated in Sec. 2, our model treats this property as a premise. However, to confirm this property,

we use the following lemma:

$$\forall id_1, id_2, pmk_1, pmk_2, t_1, t_2.\ \text{A\_HasPmk}(id_1, pmk_1)@t_1 \wedge \text{S\_HasPmk}(id_2, pmk_2)@t_2 \Rightarrow (pmk_1 = pmk_2). \quad \text{(ConfPmk)}$$

b) **Ensure that the security association keys (PTK/GTK) are fresh.** This security property states that at every run (thread tid) of the protocol it must generate a fresh PTK/GTK. We verify this property at the supplicant side through lemma

$$\forall id_1, id_2, ptk, t_1, t_2.\ \text{S\_ComputesPtk}(id_1, ptk)@t_1 \wedge \text{S\_ComputesPtk}(id_2, ptk)@t_2 \Rightarrow (tid_1 = tid_2). \quad \text{(FreshPtk)}$$

Similarly, we define Lemma (FreshGtk) for the case of GTK (omitted).

c) **Synchronise the installation of temporal keys into the MAC.** We consider the strongest authentication property from Lowe's hierarchy [17], namely, injective *agreement*. For the case of PTK, we verify that: "for each S_CommitPtk event executed by the supplicant S_{id}, the associated authenticator A_{id} executed the corresponding A_RunningPtk earlier, and for each run of the protocol there is a unique S_CommitPtk for each A_RunningPtk,"

$$\forall S_{\text{id}}, A_{\text{id}}, pars, t_1.\ \text{S\_CommitPtk}(S_{\text{id}}, A_{\text{id}}, pars)@t_1 \Rightarrow ((\exists t_2.\ \text{A\_RunningPtk}(A_{\text{id}}, S_{\text{id}}, pars)@t_2 \wedge (t_2 < t_1)) \wedge \neg(\exists S'_{\text{id}}, A'_{\text{id}}, t_3.\ \text{S\_CommitPtk}(S'_{\text{id}}, A'_{\text{id}}, pars)@t_3 \wedge \neg(t_3 = t_1))). \quad \text{(AgreePtk)}$$

Obviously, the set of parameters $pars$ must contain the value of the PTK. S_CommitPtk events are placed as late as possible on the supplicant side. A_RunningPtk events are executed as earlier as possible, when all the parameters to agree are available to the authenticator. In order to capture mutual agreement, the lemma also needs to include the case when the roles of the authenticator and supplicant are reversed. For brevity, we omit this case in our exposition, but it can be found in the source of our models [19].
As customary, authentication requires key secrecy to be asserted. We verify this using the following lemma for PTK:

$$\forall id, ptk, t_1.\ \text{S\_InstallsPtk}\,(id, ptk)@t_1 \Rightarrow \neg(\exists t_2.\ K(ptk)@t_2). \quad \text{(SecretPtk)}$$

Again, S_InstallsPtk models the primitive MLME-SETKEYS.request [3], and we require that any installed PTK is unknown to the adversary.
For GTK, we define the Lemmas (AgreeGtk) and (SecretGtk) equivalently. Moreover, we also need to capture *weak agreement* [17] of GTK in the group key handshake, through the lemma

$$\forall S_{\text{id}}, A_{\text{id}}, pars, t_1.\ \text{S\_WCommitGtk}(S_{\text{id}}, A_{\text{id}}, pars)@t_1 \Rightarrow ((\exists t_2.\ \text{A\_WRunningGtk}(A_{\text{id}}, S_{\text{id}}, pars)@t_2 \wedge (t_2 < t_1)), \quad \text{(WeakAgreeGtk)}$$

Table 1: TAMARIN results of testing properties a)-e) from the 802.11 standard and proposed property f) in Sec. 5. *No*[*Attack*] refers to (NoKrackPtk), (NoKrackGtk) or (NoDowngrade) accordingly. (✓ verified; ✗ falsified; – n/a)

Security Property	–	a) ConfPmk	b) FreshKeys		c) SynchronisedKeys					d) SameGTK	e) ConfCiphers	f) NoKeyReuse	
Lemmas	*No*[*Attack*]	(ConfPmk)	(FreshPtk)	(FreshGtk)	(AgreePtk)	(AgreeGtk)	(WeakAgreeGtk)	(SecretPtk)	(SecretGtk)	(SameGtk)	(AgreeCs)	(NoPtkReuse)	(NoGtkReuse)
PTK reinst. Figs. 4, 5	✗	✓	✓	✓	✓	✓	–	✓	✓	✓	✓	✗	✗
PTK reinst. Fig. 6	✗	✓	✓	✓	✓	✓	–	✓	✓	✓	✓	✗	✗
PTK reinst. Fig. 7	✗	✓	✓	✓	✓	✓	–	✓	✓	✓	✓	✗	✗
GTK reinst. Fig. 8	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✗
Downgrade Fig. 9	✗	✓	✓	✓	✓	✓	–	✓	✓	✓	✗	✓	✓

which includes the GTK in *pars*. As opposed to (AgreeGtk) in the 4-way handshake, the agreement in the group key handshake is not injective, because multiple retransmissions of the same GTK are allowed.

d) **Transfer the GTK from the Authenticator to the Supplicant.** We verify if the GTK received by the supplicant is the same GTK calculated and forwarded by the authenticator using lemma

$$\forall id, gtk, t_1.\ \text{S\_InstallsGtk}(id, gtk)@t_1 \Rightarrow (\exists t_2.\ \text{A\_GeneratesGtk}(gtk)@t_2 \wedge (t_2 < t_1)). \qquad \text{(SameGtk)}$$

e) **Confirm the selection of cipher suites.** We capture injective agreement of the cipher suite with Lemma (AgreeCs), similar to (AgreePtk) above, by using the cipher suite within the parameters *pars*.

We queried the lemmas defined for the above five properties in the TAMARIN models presented in Sec. 4, in order to verify them in presence of KRACK and downgrade attacks. Unexpectedly, all of the lemmas were reported as verified when KRACK attacks were present, as shown in Table 1. In the case of the downgrade attack, however, TAMARIN reported expected violation of Lemma (AgreeCs) only.

6 Proposing New Security Properties

Security Property for KRACK Attack. Section 5 clearly establishes the inadequacy of set of security properties mandated by the IEEE 802.11 standard to capture security violation by KRACK attacks reviewed in Sec. 4. Though IEEE has since addressed the issue of nonce reuse in 802.11 implementations [12], and the Wi-Fi Alliance tests the devices before certifying them for WPA2/3 [23], there is no mention of security properties being added to the standard that could capture various KRACK variants such as the ones presented by Lemmas (NoKrackPtk) and (NoKrackGtk). Accordingly, we propose an additional security property to capture such vulnerabilities:

f) Ensure that the security association keys are not used more than once.

The security property f) is encoded, using the following lemma, in Tamarin. All the KRACK attack models from Sec. 4 violate either one or both properties (See Table 1), i.e., the KRACK attacks are now captured by property f).

$$\forall id, ptk, t_1, t_2.\ \text{S\_InstallsPtk}(id, ptk)@t_1 \wedge \text{S\_InstallsPtk}(id, ptk)@t_2 \Rightarrow (t_1 = t_2). \qquad \text{(NoPtkReuse)}$$

Equivalently, we define the Lemma (NoGtkReuse) using GTK in place of PTK.

Security Property for Downgrade Attack. The downgrade attack from Fig. 9 violates property e) through the Lemma (AgreeCs). Surprisingly, the attack continue to exist even after enforcing this property as restriction. Since enforcing the agreement property on cipher suite does not stop the attack, it is violating a property not present in the standard. A detailed analysis of property e) along with the downgrade attack suggests that though the standard guarantees authentication w.r.t. other party, it does not perform agreement with itself. Accordingly, we suggest the following additional security property g), as Lemma (ValidCipherSuite), to the model of Fig. 9b that captures this attack (results omitted from Table 1 due to space constraints).

g) The cipher suite that the authenticator started with is the cipher suite that the authenticator finishes with, and is the strongest one from the available choices.

As expected, the downgrade attack from Sec. 4 is captured by property g), which is encoded in Tamarin using lemma

$$\forall tid, cs_1, cs_2, t_1, t_2.\ \text{A\_SentMsg3}(tid, cs_1)@t_1 \wedge \text{A\_Starts}(tid, cs_2)@t_2 \Rightarrow (cs_1 = cs_2). \qquad \text{(ValidCipherSuite)}$$

To verify that our proposed security properties f) and g) corresponds to respective attacks, we fix the respective Tamarin models of Sec. 4 by enforcing (NoPtkReuse), (NoGtkReuse) and (ValidCipherSuite) as *restrictions*.

On testing the security properties from Sec. 4, i.e., Lemmas (NoKrackPtk), (NoKrackGtk), and (NoDowngrade), Tamarin verifies them in the fixed model, proving that the proposed security properties are successful in stopping these attacks.

7 Verifying the Mitigations to the Models

Finally, we fix the KRACK models, from Sec. 4, making sure that they follow the newly proposed security property f), i.e., disconnect if there is an attempt to install with the same PTK or GTK, and then execute the model again. After the fix, both of the attack lemmas, i.e., Lemmas (NoKrackPtk) and (NoKrackGtk),

along with the security properties (NoPtkReuse) and (NoGtkReuse) are verified. The absence of the attack, with the new security properties verified, shows the validity of the proposed fix. This result is also a verification of the proposed countermeasure for KRACK by [22].

Similarly, the downgrade attack from Fig. 9 can be easily detected at the supplicant side [20], and can be stopped if the authenticator implementation disallows the change of cipher suites mid-handshake. Accordingly, we fix the model ensuring that it rejects a connection where the authenticator does not start and finish with the same cipher suite. After fixing it, Tamarin reports the attack Lemma (NoDowngrade) as verified, i.e., the downgrade attack no longer exists, and that the mitigation is valid. Both the fixed Tamarin models, of KRACK and downgrade attacks, are publicly available at [19].

8 Conclusion and Further Work

We have presented formal models of various KRACK attacks on the IEEE 802.11 4-way handshake and group key handshake, and downgrade attacks on implementations of the 4-way handshake. Using the automatic verification tool Tamarin, we verify all the security properties of the 4-way handshake mandated by the 802.11 standard, in the presence of KRACK and downgrade vulnerabilities. We find that KRACK attacks do not violate any of the required security properties. We conclude that the set of properties is inadequate to capture these attacks. Using a novel approach, we propose additional security properties to be added to the 802.11 standard, enabling it to capture them. We also demonstrate that enforcing these security properties in our model successfully stops these attacks. Accordingly, we fix the models with countermeasures to mitigate the attacks and verify all the security properties, providing a formal proof of correctness of the recommended countermeasures. Our novel technique can strengthen protocol specifications, by testing the adequacy of the set of required security properties against known or newly discovered attacks, and by augmenting them with new properties, if required. For future work, we would like to extend it to other use cases, i.e., to test the set of required security properties for other protocols against known attacks on them.

Acknowledgements

We would like to thank Robert Künnemann, Chris McMahon Stone and Mathy Vanhoef for useful discussions. We would also like to thank the anonymous reviewers for their insightful comments and suggestions. This work was partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 779391 (FutureTPM).

References

1. IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. IEEE Std. 802.11-1997 (Nov 1997)

2. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6. IEEE Std. 802.11i-2004 (Jul 2004)
3. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. IEEE Std. 802.11-2016 (Dec 2016)
4. IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control. IEEE Std. 802.1X-2020 (Feb 2020)
5. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. ACM SIGPLAN Not. **36**(3), 104–115 (Jan 2001)
6. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Proc. Int. Cryptol. Conf. (CRYPTO). LNCS, vol. 773, pp. 232–249. Springer, Santa Barbara, CA (Aug 1993)
7. Blanchet, B., Smyth, B., Cheval, V., Sylvestre, M.: ProVerif 2.02: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial (Jul 2020)
8. Cremers, C.: On the protocol composition logic PCL. In: Proc. ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS). pp. 66–76. Tokyo, Japan (Mar 2008)
9. Cremers, C., Kiesl, B., Medinger, N.: A formal analysis of IEEE 802.11's WPA2: Countering the kracks caused by cracking the counters. In: Proc. USENIX Security Symp. (USENIX Security). pp. 1–17. Virtual event (Aug 2020), to appear
10. Datta, A., Derek, A., Mitchell, J.C., Roy, A.: Protocol composition logic (PCL). Electron. Notes Theor. Comput. Sci. **172**, 311–358 (Apr 2007)
11. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Trans. Inf. Theory **29**(2), 198–208 (Mar 1983)
12. Harkins, D., Malinen, J.: Addressing the issue of nonce reuse in 802.11 implementations. Available at `https://mentor.ieee.org/802.11/dcn/17/11-17-1602-03-000m-nonce-reuse-prevention.docx` (Oct 2017)
13. He, C., Mitchell, J.C.: Analysis of the 802.11i 4-way handshake. In: Proc. ACM Workshop Wirel. Secur. (WiSe). pp. 43–50. Philadelphia, PA (Oct 2004)
14. He, C., Sundararajan, M., Datta, A., Derek, A., Mitchell, J.C.: A modular correctness proof of IEEE 802.11i and TLS. In: Proc. ACM Conf. Comput. Commun. Secur. (CCS). pp. 2–15. Alexandria, VA (Nov 2005)
15. Joux, A.: Authentication failures in NIST version of GCM. Pub. C. to NIST (2006)
16. Kremer, S., Künnemann, R.: Automated analysis of security protocols with global state. J. Comput. Secur. **24**(5), 583–616 (Nov 2016)
17. Lowe, G.: A hierarchy of authentication specifications. In: Proc. IEEE Comput. Secur. Found. Workshop (CSFW). pp. 31–43. Rockport, MA (Jun 1997)
18. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Proc. Int. Conf. Comput.-Aided Verif. (CAV). LNCS, vol. 8044, pp. 696–701. Springer, St. Petersburg, Russia (Jul 2013)
19. Singh, R.R., Moreira, J., Chothia, T., Ryan, M.D.: TAMARIN prover models of attacks on the 802.11 4-way handshake to verify security properties (source code and proofs) (2020), available at `http://people.du.ac.in/~rrsingh/wpa2models`
20. Stone, C.M., Chothia, T., de Ruiter, J.: Extending automated protocol state learning for the 802.11 4-way handshake. In: Proc. European Symp. Res. Comput. Secur. (ESORICS). LNCS, vol. 11098, pp. 325–345. Springer, Barcelona, Spain (Sep 2018)
21. Vanhoef, M., Piessens, F.: Predicting, decrypting, and abusing WPA2/802.11 group keys. In: Proc. USENIX Secur. Symp. pp. 673–688 (2016)
22. Vanhoef, M., Piessens, F.: Key reinstallation attacks: Forcing nonce reuse in WPA2. In: Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CSS). pp. 1313–1328. Dallas, TX (2017)
23. Wi-Fi Alliance: Security update october 2017. Available at `https://www.wi-fi.org/security-update-october-2017` (Oct 2017)