Information Security Final Project

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

古佳傢 60947005S

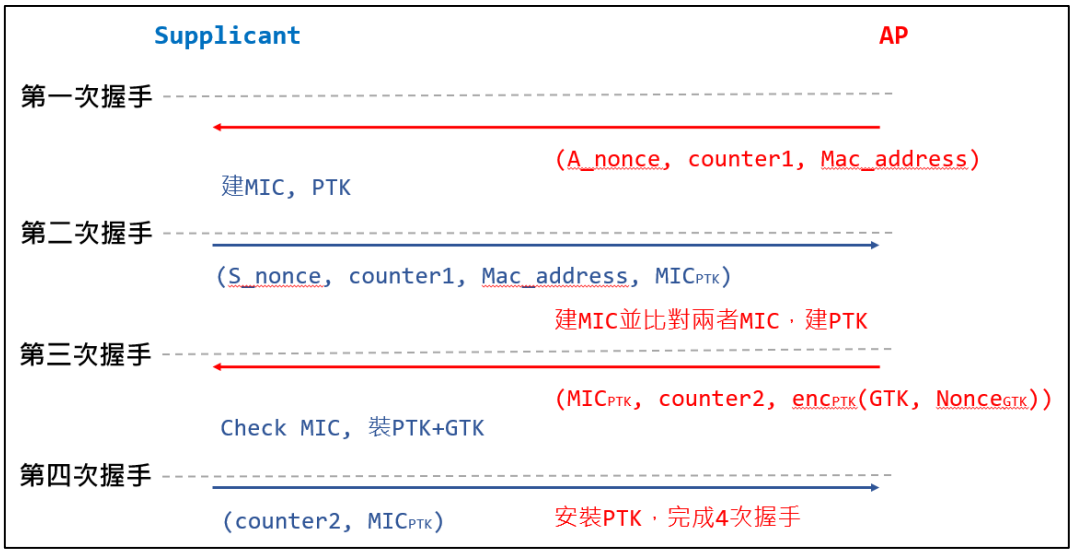
鄭博升 60947038S

研究動機：

這次期末報告，起初想要找一篇可以實作的 paper 來實際玩玩看，在看到這篇 paper 的標題是對 wpa2 的 krack 攻擊分析時，基於實作過中間人攻擊的作業後對此類型的題材非常有興趣，於是毫不猶豫的決定可以來研究看看，由於我們在作業中嘗試進行中間人攻擊時的先備知識的不足，很多東西都是從頭開始查到尾，然而老師上課也有介紹到 wpa2 以及 krack 攻擊，讓我們對於 wpa2 稍微有了一些認識。但在讀完後才知道這篇 paper 其實是針對目前的 wpa2 協定進行詳細分析、證明並且設計一個安全實用的模型，其中驗證與證明的部分是使用第一作者所設計的 Tamarin Prover 來進行，實際下載作者的模型，並將其導入 Tamarin 後發先其內容驚為天人的複雜，但也充滿了規則，每個元件都在重重 lemma 下一步一步進行自動驗證。在經過這篇 paper 探討後，我們也藉此更了解目前市面上我們使用的 wpa2 是如何運作的，並且存在著哪些風險需要注意，最後雖然沒有完整的實作可以讓我們嘗試，但我們透過作者設計的模型在 Tamarin 上的驗證中學到很多新知。

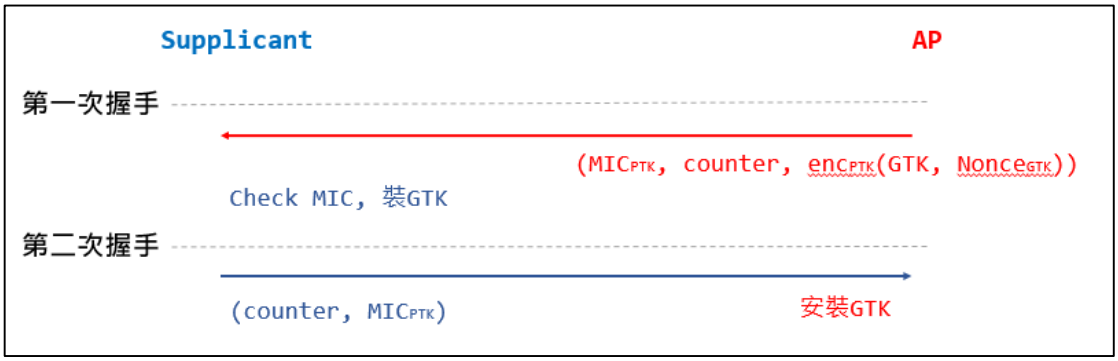
背景介紹：

Four-way Handshake



當 supplicant 想要與 AP 端建立連線，會先輸入 wifi 密碼，此時會產生 PMK(由 WIFI 密碼、SSID 經過 hash function 所構成)，接著 AP 會對 supplicant 發送 A_nonce、counter1、Mac_address，此為第一次的握手，而 counter 是作者設計的 Replay Counter 利用 receiver 必須回覆和 sender 訊息中所挾帶的 counter 數值相同的數值來避免 replay attack，當 supplicant 收到訊息後，即可透過雙方的 Mac_address 建立 MIC，並透過雙方的 nonce 與 PMK 建立 PTK，而 supplicant 會將 S_nonce、counter1、Mac_address、MIC 回傳給 AP 端，完成第二次握手，在 AP 收到後，也會用雙方的 MAC_address 建立 MIC，並比對傳來的 MIC 是否一致，來確保封包的完整性，接著也會建立 PTK，並傳送 MIC、counter2 以及被 PTK 加密的 GTK，做為第三次握手，supplicant 則會在確認 MIC 正確後，安裝 PTK 與 GTK，並回傳 counter2、MIC，告知 AP 安裝完成後，此時 Four-way handshake 就完成了。

Group-key Handshake



GTK 是一個群組暫時金鑰，主要用於區網內的廣播訊息加密，因此在一段時間或是有 supplicant 請求更換 GTK 時，AP 端會透過 Group-key Handshake 的方式來發送新的 GTK 給所有連線中的 supplicant，運作模式類似於 four-way handshake 中的第三與第四次握手。

實驗模型：

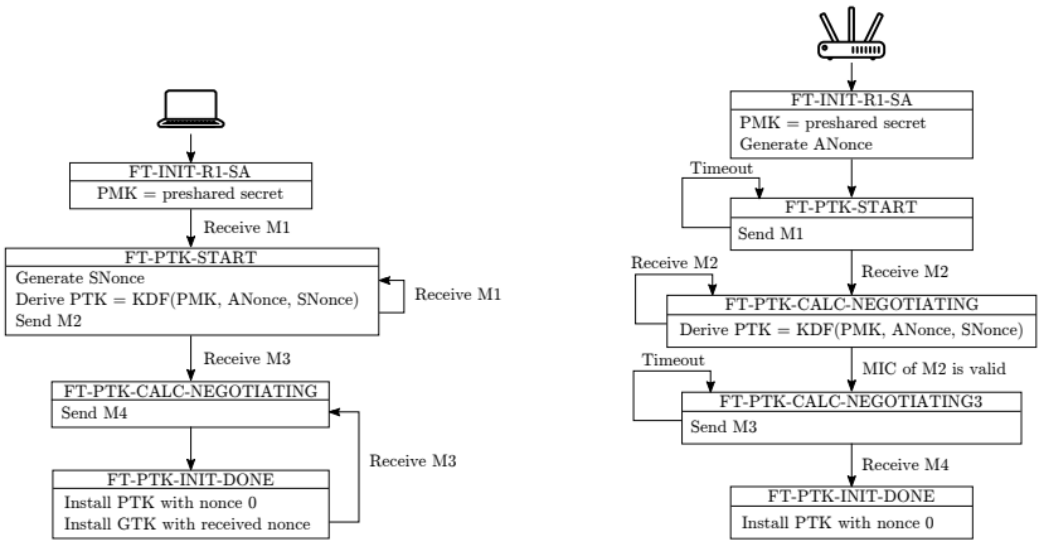
作者透過 Tamarin Prover 建立 four-way handshake 與 group key handshake 等模型，並透過此軟體進行自動驗證。

在 wpa2 中，若要實作 four-way handshake 與 group-key handshake 須滿足以下規範

- 實作 four-way handshake 需使用至少兩台的狀態機，一台為 supplicant，另一台為 AP
- 實作 group-key handshake 需使用至少兩台狀態機，一台為 supplicant，另一台為 AP
- 一台狀態機專門用於產生新的 group key

Four-way handshake

Four-way handshake 狀態轉換圖



(a) Supplicant State Machine.　　(b) Authenticator State Machine.

在 four-way handshake 中，當傳送完 M1 以及 M3 後，遲遲沒有收到 supplicant 的訊息，則會造成 timeout，導致 AP 端重新傳送一次原訊息，這也造成了之後的 krack 問題。

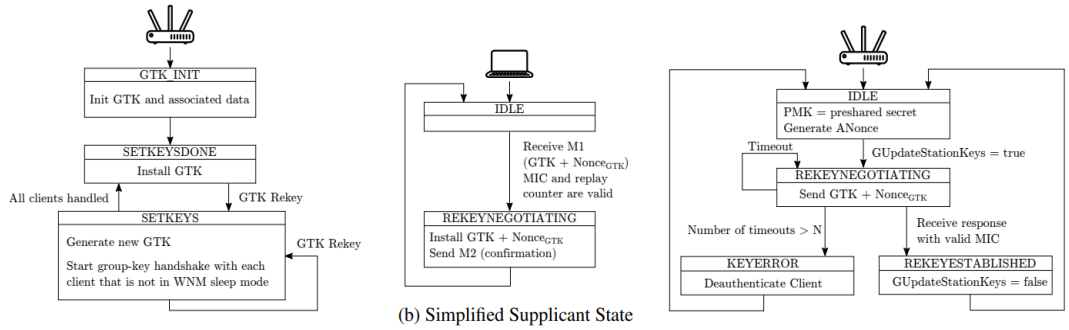
腳本 1 作者展示了 supplicant 端，從 FT-PTK-START 狀態轉換到 FT-PTK-CALC-NEGOTIATING 狀態的 Tamarin 程式碼。SuppState 代表現在的狀態在 PTK_START，並且每個 STA 都有一個獨一無二的 suppThreadID，"~"代表之前生成的東西，InEnc 代表從網路中接收到的資訊，SuppRcvM3 在此 model 中用於驗證 lemmas，SuppSeesCtr 用於通過限制對 replay counter 機制進行建模，Eq 用於確保此 message 的 MIC 是 valid

腳本 1: FT-PTK-START 狀態轉換到 FT-PTK-CALC-NEGOTIATING 狀態

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

Group-key handshake

Group-key handshake 狀態轉換圖



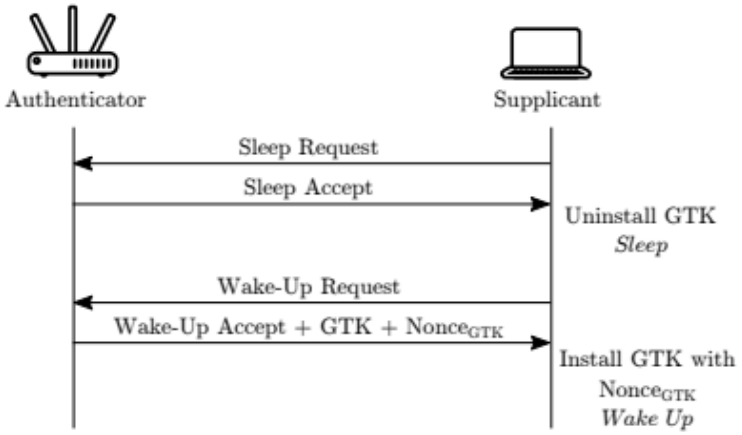
(a) Simplified Global Authenticator State Machine for GTKs.

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

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

在 group-key handshake 中，(a)用於生成新的 GTK，(b)與(c)則用於模擬 group-key handshake，每個 supplicant 可透過發送型態為 group bit 的 Request bit 設為 1 的 EAPOL-key frame 觸發 group key handshake，AP 也可在有人斷線或認證失敗的時候發起 group-key handshake，以及從 sleep mode 醒來的 supplicant 也需要進行 group-key handshake 來拿到當前使用的 GTK。在此 model 中，four-way handshake 與 group-key handshake 無法同時發生，若同時都需要進行，則以 four-way handshake 優先，並且在 handshake 過程中的 replay counter 是相對第一個訊息中各自的 handshake replay counter，因此這兩種 handshake 也不該同時進行。

WNM Sleep Mode



此模式允許 supplicant 進入睡眠模式來節省能源，此時會將 GTK 解除安裝，並不參與 group-key handshake，此處作者為了使模型更加具有一般性，讓 WNM 相關的通訊模式只要在建立好 PTK 後，可以和其他 handshake 同時發生。另外提到再進入睡眠模式後必需將原先安裝的 GTK 刪除，才不會有後續 Message Queue 的問題產生。

Message Queue

在真實情況中，很有可能發生訊息已送出，但卻還在 Message queue 排隊的現象，所以假定在一種情景是 supplicant 申請新的 GTK 但有為時間的關係訊息還在 Queue 中排隊，但在尚未申請完成前，如果 supplicant 使用 GTK 加密時就可能造成使用舊的 GTK 的情況發生，因此作者設計一個腳本 2 來實作 Message Queue，並透過 Tamarin Prover 來驗證這類的漏洞不會在他的系統發生。在此模型中，每個 supplicant 都有獨立的一個 message queue，和自己的 threadID，圖中的 OutEnc 代表將訊息內容放入 message queue 中。

腳本 2：Message Queue

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

WPA2 加密與傳送

WPA2 的訊息加密採用 AESCCMP 來加密，腳本 3 在尚未建立 PTK，訊息將直接以明文方式傳送，腳本 4 在建立 PTK 後，則是會有一個初始化的 nonce 搭配 AES 加密產生 key stream，再與明文做 XOR，並且每次加密完後 nonce 的值會一直往上加，以避免 nonce reuse 而產生 Krack 的漏洞。

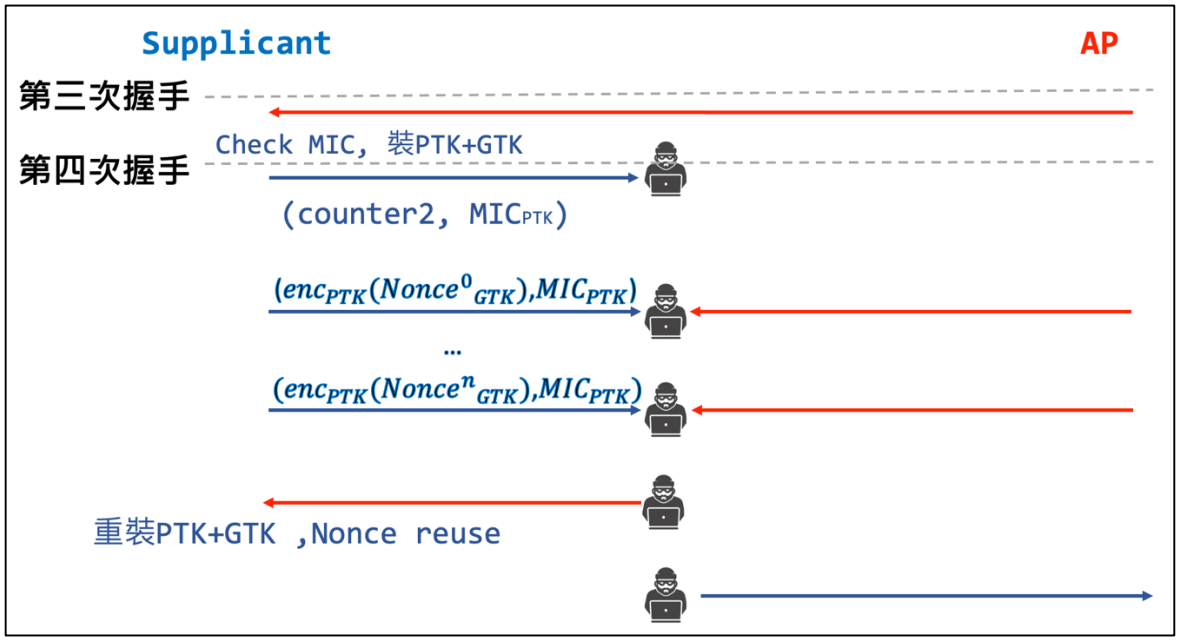
腳本 3：PTK 尚未安裝

$$\begin{aligned}&[\ \mathsf{OutEnc}(message,\sim senderThreadID,\sim msgID)\ ]\\ &-[\ \mathsf{SendMessage}(\sim senderThreadID,\sim msgID)\ ]\to\\ &[\ \mathsf{Out}(message)\ ]\end{aligned}$$

腳本 4：PTK 安裝後

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

Krack:



1. 一般情況下的 Krack 發生在第四次握手時，supplicant 在傳送 ACK 給 AP 時，如果中間攻擊者劫持著這段訊息，則 AP 在未收到 supplicant 的 ACK 時會覺得 supplicant 還未正確安裝 PTK+GTK，於是將會再等待某一個時間 timeout 之後，重新發送第三次握手的訊息給 supplicant，然而 supplicant 在收到重複的第三次握手的訊息時，就算已經安裝好 PTK+GTK 了，依然會因為是既有的設計而重複安裝 PTK+GTK 的動作，經過反覆操作，造就了中間攻擊者可以在中間進行控制，因為當 supplicant 在安裝完 PTK+GTK 後，會根據 CCMP 加密流程中的取用 Nonce 來和 TK 進行 AES 加密的這個動作，在未修補的情況下 Nonce 都是經由初始值零至某個上限值，而且 TK 都是同一個的情況下，如果中間攻擊者可以在經過 n 次後取得重複的 Nonce，那就代表 nonce reuse 的情況發生，這時候就會非常像 Broken one-time-pad 的情況，同一個 key stream 重複使用造成中間攻擊者可以利用公式 1 的作法，即使不需要知道完整的 key stream 也可以暴力破解出敏感資訊。

公式 1: Broken one-time-pad

$$\begin{cases} c1 = m1 \oplus PRG(key) \\ c2 = m2 \oplus PRG(key) \\ c1 \oplus c2 = m1 \oplus PRG(key) \oplus m2 \oplus PRG(key) \oplus = m1 \oplus m2 \end{cases}$$

2. 在發生最壞的情況下，如果發生 Krack 且中間攻擊者可以透過 nonce reuse 的計算找出完整的 Key Stream 那不就代表作者可以再往後

supplicant 所傳輸的訊息直接被洩露，如圖表 1 所示。作者針對 worst case design 的情況在 Tamarin 上並透過 nonce reuse 不會發生的 lemma，來確保這個最壞的情況不會發生在作者所設計的模型中。

腳本 5：worst case design

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

圖表 1：worst case 的假設情況，Key Stream 被完整洩漏

worst case		
Nonce	Key	Key Stream
0000	f4b7	$Key\ Stream^{0}$
0001	f4b7	$Key\ Stream^{1}$
0002	f4b7	$Key\ Stream^{2}$
..	f4b7	..
ffff	f4b7	$Key\ Stream^{65535}$
0000	f4b7	$Key\ Stream^{0}$

3. Krack 如果發生在 Linux & Android 2.4/2.5 version system 時會發生一個特殊的情況，工程師在設計時因為沒想到會發生 Krack，於是在 supplicant 安裝完 PTK+GTK 後立刻就把 TK 歸零，從 wpa 未修正前的開源程式碼找到確實有一段是當 TK 不再需要時就將其歸零的設計，這個動作導致後續如果發生重裝 PTK+GTK 這件事情的時候，supplicant 會使用全為零的 TK 來生成 Key Stream，於是往後 supplicant 在通訊時都會以這個不安全的 Key Stream，所以通訊一直都處在極大的風險之中，但這個問題其實並沒有被作者提及，而且問題也在軟體更新後解決。

程式碼 1

```
/* TK is not needed anymore in supplicant */
os_memset(sm->ptk.tk, 0, WPA_TK_MAX_LEN);
```

Prevent Krack：

Replay counter rule：針對 CCMP 加密時的 nonce 做一個簡單的設計，讓 nonce reuse 這件事情不會發生。作者採用起始值是一個非零的任意數值以 nonce 稱之，並確保每次在進行加密的過程中所使用的 nonce 都比前一個值還要大至少一個單位的規則來設計，並且引用 Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA 這篇 paper 的策略: supplicant 在安裝完 PTK+GTK 後不可擅自更改或重置 CCMP 加密所使用的 nonces，換句話說只有你在重新進行 four-way handshake 時才會將 nonce 重置成某個任意值。再透過設計在 Tamarin 的檢查腳本，每次都對於下一個加密所使用的 nonce 檢查必定是在大於某個數值 x 的情況下，才能確保 nonce reuse 的情況不存在。

腳本 6：Replay counter rule

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

腳本 7:nonce reuse check rule

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

KrOOk Vulnerability：

ESET 2020 年在資安大會上發佈的關於博通和聯發科製作 wifi 晶片上的漏洞，其漏洞在於針對 wifi 解離時緩衝區中的封包因為時間的關係還未被傳送完全，但此時卻因為晶片設計會將暫時金鑰設成零來對那些封包進行加密傳輸，所以中間攻擊者如果可以控制 supplicant 和 AP 之間解離時，中間攻擊者就可以在敏感的時間取得敏感的資訊，然而這個漏洞在發布後幾個月博通及聯發科都分別發佈 wifi 驅動程式來修補這個漏洞，其中修補的方式並未在網路上公佈，這邊推測可能是當 supplicant 和 AP 發生 wifi 解離的狀況時，針對那些還在緩衝區的封包就不再進行任何加密傳輸，而是直接丟棄，就可以解決這個問題。

Conclusion:

作者針對模型的安全性以用四大項目分別來證明其是否安全，其中 PTK、GTK、Four-way handshake 更是以 supplicant 和 AP 端兩個不同的角度來進行分析：

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

PMK:

PMK 由 passphrase、SSID 經過 hash function 的結果，由於 PMK 並未透過網路直接傳輸，而且在加密上野並非直接使用 PMK 來做加密，所以對於 PMK 的安全只要 wifi 密碼未被洩露，那 PMK 的安全就無須考慮。

PTK:

考慮到 PTK 是由 PMK、Anonce、Snonce 經過一系列運算生成的，所以在只要在 PMK 不被洩漏的情況下那 PTK 也應該就是安全，並且作者已經利用 reply counter 證實他的模型不會有 nonce reuse 問題，所以 PTK 是安全的。

GTK:

因為 GTK 是由 PTK 加密而來，所以如果 GTK 是不安全，那就表示 PTK 已經被洩漏，又因為 PTK 是由 PMK、Anonce、Snonce 經過一系列運算生成的，因此若 PTK 被洩漏那表示 PMK 也被洩漏，但不失一般性假設 PMK 不會被洩漏，而且 model 證明 nonce reuse 不可能會發生所以 PTK 也是安全的且 GTK 也是安全。

Four-way handshake:

由於 Four-way handshake 只會執行一次，且 Anonce 與 Snonce 每次生成都不一樣且傳輸過程中都只有一個，基於這樣的前提下作者在 Tamarin 中所設計的一系列 lemma，來證實 Four-way handshake 有滿足 injective agreement。

總結來說作者針對 WPA2 在基於 Krack 的攻擊為其研究動機提出了一個能抵擋各種可能出現攻擊的模型架構，並透過工具 Tamarin 來進行理論和數學上的驗證，也確實驗證抵擋包括 Krack 各種假設攻擊。