ILLINOIS INSTITUTE OF TECHNOLOGY

Illinois Institute of Technology

Master's degree of Information Technology and Management:

Computer's security

Research Thesis

Approach to a possible misunderstood of the 802.11 standard while developing the Key Reinstallation attack exploit.

PABLO JUSUÉ FERNÁNDEZ

Project director:

Jeremy Hajek

Contents

List of Figures

Abstract

The 802.11 standard is world wide known and use in every communication.This standard it is also commonly known as the Wifi protocol. The standard and its family of protocols are defined for IP communication between different users using IP devices. But in this project I am going to study the security protocol known as WPA2 which is used for establishing a secure communication between the user device and the access point. WPA2 is the newest security protocol implemented in the 802.11 standard for the key exchange between two nodes. This exchange it is done by sending four messages, this is the reason why this is called the 4-way handshake. The key reinstallation attack it was developed by Mathy Vanhoef of imec-DistriNet, and allows the attacker to read the information which was previously assumed to be encrypted. During this project I have analyzed two approaches for possibles misunderstood of the standard when performing this attack

802.11 estandarra mundu guztian ezaguna da eta komunikazio guztietan erabiltzen da. Estandar hau WiFi protokolo izenaz ere ezagutzen da. Estandar hau eta haren protokoloen familiak IP gailuen erabiltzaile ezberdinen arteko komunikazioetarako zehaztu dira. Hala ere, proiektu honetan, WPA2 izeneko segurtasun-protokoloaren estudioa egingo dut. Protokolo hau erabiltzailearen gailuaren eta sartze-puntuaren arteko komunikazio segurua eratzeko erabiltzen da.WPA2 802.11 estandarrean inplementatutako segurtasun-protokolorik berriena da, bi nodoren arteko gako-trukea egin dadin. Truke hau lau mezu bidaliz burutzen da, eta, horren ondorioz, 4-way handshake izena ere hartzen du. Gakoen berrinstalazio erasoa Mathy Vanhoef-ek garatu zuen, imec-DistriNetekoa, eta, horren bidez, erasotzaileak enkriptatuta izan beharko litzatekeen informazioa irakur dezake. Proiektu honetan zehar, eraso bat egiterakoan gerta daitezkeen gaizki-ulertuen bi hurbilketa aztertu egin dira.

El estandar 802.11 es utilizado en todas las comunicaciones a nivel global. También se le conoce de manera común y gracias al apoyo de la wifi aliance como el estandar Wifi. El estándar y su familia de protocols se definen para comunicaciones IP entre los distintos usuarios que usan dispositivos IP, pero en este proyecto se va a estudiar el protocolo de seguridad perteneciente a esta familia llamado WPA2 que se utiliza para establecer comunicaciones seguras entre los dispositivos de usuario y los puntos de acceso. WPA2 es el actual protocolo de seguridad que implementa el estándar 802.11 para el intercambio de claves entre dos nodos. Este intercambio de claves se realiza enviando 4 mensajes, por esa razón se le llama "4-way handshake". El ataque de reinstalación de clave (KRACK), que ha sido desarrollado por Mathy Vanhoef de imec-DistriNet permite al atacante leer la informacion que debería haber sido encriptada. Durante este proyecto se van a analizar dos posibles errores de interpretación realizados por Mr. Vanhoef a la hora de realizar este ataque.

1 Introduction

The life is now based on internet. Most of the communication are done using different IP devices or different IP application.However, regular users sometimes do not get involve with the security needed for this applications to work. It is an important amount of knowledge in cryptography and communications for understanding the security protocols which are implemented in a regular communication for protecting the data from unauthorized users.

In a regular communication it is important to protect the confidentiality and the integrity of the data transmitted by the users involved. First it is important to understand what is the definition of the concepts confidentiality and integrity. William Stallings and Lawrie Brown define as clear as possible this concept in their book ”Computer Security Principles and practice”[1]

- Data confidentiality: Assures that private or confidential information is not made available or disclosed to unauthorized individuals.
- Data integrity:Assures that information and programs are changed only in a specified and authorized manner.

In every communication these concepts must be applied so we can assure that the communication is secure. For this study we are going to analyze the data confidentiality during a 802.11 communication. The confidentiality is reached by using cryptography tools and methods to encrypt the data which is sent in the communication.

One of the main factors to reach the confidentiality in a communication is the use of a private key. In every scheme of encryption, there is a key that must only be known by the two user which are communicating. This key must be private during the whole communication because if an unauthorized user access to this key, he/she will be able to access to the whole raw data which is been sent.

1.1 KRACK exploit

The KRACK[2] exploit attacks the confidentiality of the 802.11 standard by attacking a vulnerability found in the cryptographic protocols of key exchange, which allows the attacker to reinstall an already-in-use key. This is done by re-sending a message of the 4-way handshake to the user device so it thinks that the key hasn't been installed yet and it re-install it again. This vulnerability can only be exploited in some systems because, as it is stated in the KRACK paper, some of the OS are not implementing correctly the 802.11 standard.

1.2 Hypothesis of the project

During this project and based in the KRACK exploit paper and the 802.11 standard, I have investigated two different hypothesis of a possible misunderstood of the 802.11 standard done in the KRACK exploit. The first hypothesis is based in the statement that the implementation of the 802.11 standard is incorrectly done by Windows and Apple and that is why they are not vulnerable to this exploit. The second one is based in a wrong implementation of the standard by Windows and Apple systems but which can be exploited too.

The first hypothesis was developed by Mr. David Rose Sr. Network Engineer at Illinois Institute of Technology. The second one was developed by my, Pablo Jusue Fernandez, graduate student at the Illinois Institute of Technology.

2 Goals of the project

During this project I am going to perform the analysis of two hypothesis and to study the 802.11 standard for evaluating the correct implementation of the standard which have been done by the different systems and trying to find possible flaws in the cryptography system and in the Key exchange process, also known as 4-way handshake. The steps I am going to follow are the next ones:

- Install the KRACK[2] scripts to perform a POC and learn how the attack is implemented and how this attack could be patched in the vulnerable systems.
- Study the 802.11 standard for learning how the key exchange is done and how each message of the 4-way handshake should work either in the supplicant or the authenticator.
- Analyze the first hypothesis using the information obtained from the standard and from the attack and check if the implementation done by the vulnerable systems is correct or not.
- Analyze the second hypothesis and check a possible configuration done by the OS and Windows systems when implementing the 802.11 standard.
- Looking for a possible solution for the KRACK vulnerability in the different systems which are vulnerable.

3 IEEE 802.11 security

In the 802.11 networks two classes of security algorithms are defined.

- Pre-RSNA algorithms: The pre-RSNA security mechanism are deprecated and should be removed from the current implementations of the standard.
- RSNA algorithms:

3.1 Pre-RSNA

The base security mechanism used in the Pre-RSNA[3] is the Wired Equivalent Privacy (WEP[4]). The first version WEP-40 was defined, using a 40-bit key, for protecting the confidentiality of the communication between two different user from unauthorized user who wanted to access to this information.

A 40-bit key is to weak for protecting the communications from the different crypto-text attacks, so this algorithm also has been with a 104-bit key, known as WEP-104. The algorithm didn't change with this update in the key length, so when we refer to the WEP mechanism could be either WEP40 or WEP104.

Because WEP only uses keys for encryption it only covers the confidentiality of a communication, not the authentication or the integrity of it. During the encryption process WEP uses two different types of keys:

- Key-mapping keys
- Default keys

A key-mapping key is a key assigned to a communication between a pair of users. Basically it is a private key system for encrypting the MPDU'S which are transmitted by both users. If a key-mapping key is not defined between a pair of users, for example this will happend if it is the first time they are communicating one with each other, it will be used the default keys

The default keys can be used in two different ways:

- Using the default key to encrypt a key-mapping key so this one can be used to encrypt the communication
- Using the default key to encrypt the information between the pair of users. If this option is choose the use of the default key will be indicated in the key-id field of the header of the WEP MPDU.

This header can be seen in the following image obtained from the 802.11 standard of 2016

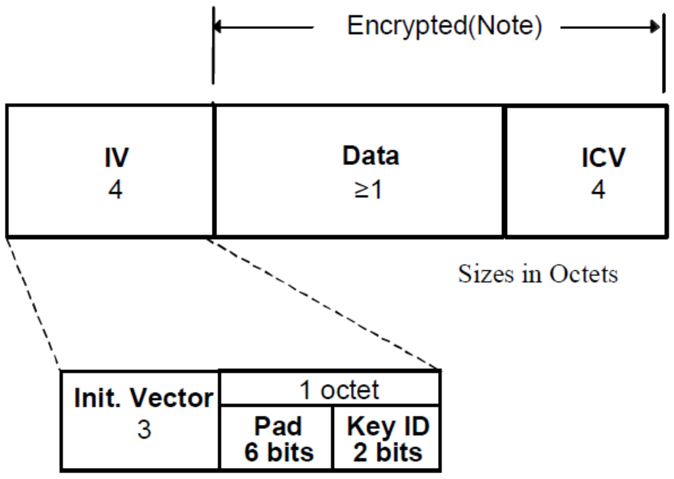


Figure 3.1: Construction of expanded WEP MPDU from the 802.11 standard (2016 version)

In the standard the Key ID is defined as a 2 bit array which means there are 4 different default keys which can be used and will be referenced here by using the values 0,1,2,3.

If for the communication a Key-mapping key it is being used the Key ID field will be reserved.

The WEP encryption algorithm shall use the ARC4[5] stream cipher, also known as RC4, from the RSA security, INC as its encryption or decryption algorithm. This is one of the problems of the WEP mechanism because multiple vulnerabilities have been found in the ARC4 which directly affects the WEP mechanism security due to the encryption and decryption process is exposed.

The per-MDU key used by the WEP mechanism is called *seed* and it is the concatenation of the encryption key to an Initialization vector (IV). An example of this is that in the WEP-40 the 40 bits represented in the name, actually are referencing the bits 24-63 of the seed and are the encryption key.

3.1.1 PRE-RSNA authentication

As I have explained before the WEP mechanism doesn't offer an authentication method.

First I need to explain some of the concepts which I will talk about in the following possibilities. A DMG STA is a station which works in a channel frecuency starting at 45GHz. The initials means *Direct multi-gigabit*.

A IBSS is a *basic service set*, BSS, that forms a self-contained network, and in which no access to a distribution system is available. Now with this concept defined I am going to explain the three different possibilities for the authentication

For authentication there are three possibilities:

- A non-DMG STA needs to complete an 802.11 authentication exchange prior to association
- A DMG STA not in an IBSS needs to complete the authentication exchange when other than the Open System authentication algorithm is requested and shall not perform the authentication exchange using the Open System Authentication algorithm
- An IEEE 802.11 authentication exchange is optional in a IBSS

3.2 RSNA confidentiality and integrity protocols

There are three confidentiality and integrity protocols which are:

- Temporary key integrity protocol (TKIP)
- CTR with CBC-MAC protocol (CCMP)[6]
- GCM protocol (GCMP)

These protocols are important to understand because there are the ones which the KRACK exploit attacks.

3.2.1 Temporary Key Integrity Protocol (TKIP)

TKIP is a cypher suite which modifies WEP doing the following operations:

- It calculates a message integrity code, which is called MIC, from the MSDU and attach it to the MSDU data prior fragmentation. The receiver will verify the MIC value, after defragmenting the MPDU's into the MSDU and if the value is not the same as the one calculated by the receiver, this one will discard the MSDU, because the integrity was vulnerable by unauthorized user.
- Because this implementation canť protect against forgery it has other countermeasures. In each MPDU TKIP add a sequence number. This sequence number is called TKIP sequence counter (TSC)

In the following image there is a scheme of how the TKIP protocol works. This scheme was taken from the 802.11 standard of 2016.

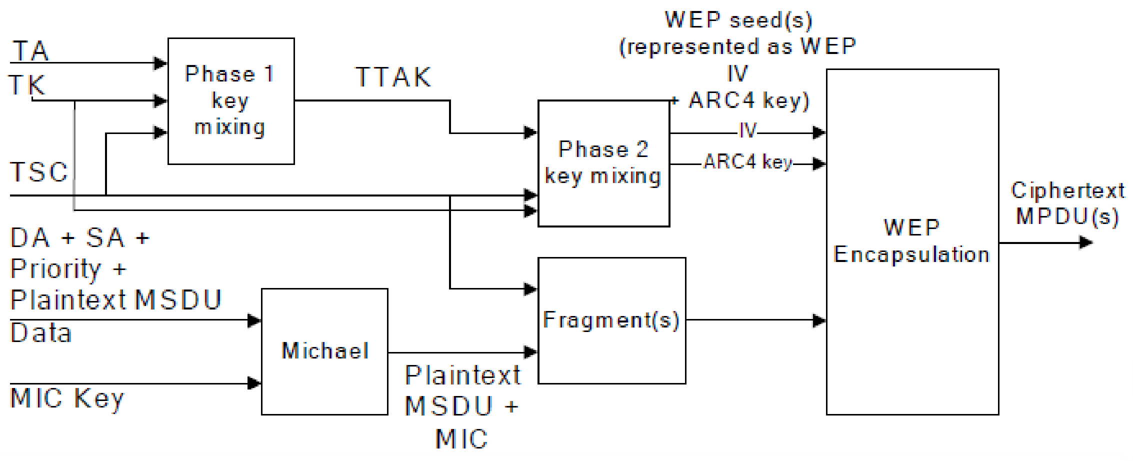


Figure 3.2: TKIP encapsulation from the 802.11 standard (2016 version)

The TKIP MSDU reuses the MSDU from WEP but it adds to it 12 octets. The first four octets are used for adding the WEP seed obtained from the Phase 2 key mixing, which can be seen in the previous scheme, and the other 8 octets are used for adding the MIC value.

The TSC is composed by 6 octets from TSC0-TSC5 where TSC5 is the most significant octet. The first two octets TSC0 and TSC1 goes from 0xFFFF-0x0000 and when it rolls over the upper 32 bits (4 octets) increments by one.

3.2.2 CTR with CBC-MAC protocol (CCMP)

The CCMP protocol is based on the CCM of AES encryption algorithm. For defining CCM as easier as possible I will define it *as a block cypher mode which use a symmetric key and provides confidentiality using the counter mode. It also provides authenticity and integrity by using the message authentication code.* This part of the protocol which ensures integrity and authentication it is called CBC-MAC because it is a MAC code applicated to a Chain-block-cypher (CCM).

There are two implementations of the CCMP protocol.

- CCMP-128
- CCMP-256

The difference between this two implementation is the value assigned to the CCM parameters:

- M indicating the length of the MIC
  - 8 for the CCMP-128
  - 16 for the CCMP-256
- L indicating that the length field is 2 octets which is enough for the largest MPDU.
  - It is 2 for CCMP-128 and CCMP-256

When CCMP is used the MPDU has the following format:

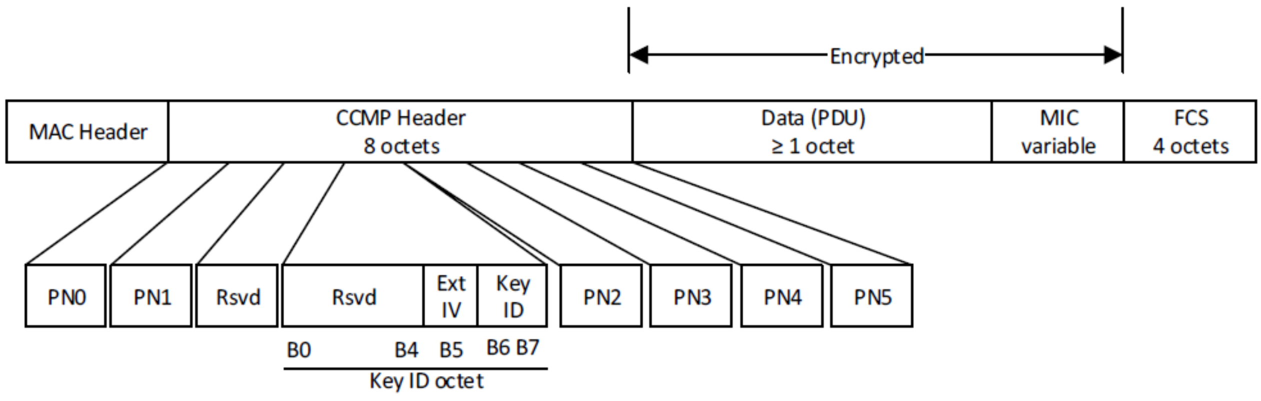


Figure 3.3: CCMP MPDU from the 802.11 standard (2016 version)

In the CCMP header there are 6 octets used by the Packet number(PN). It use is the same as the TSC in TKIP. Then there is the ExtIv which is the B5 of the Key ID octet indicates, as in TKIP, that the WEP MPDU header extends by a total of 8 octets compared to the 4 of WEP, so it is always set to 1.

4 Resources

For developing this project I will need different hardware and software devices:

4.1 Hardware devices

For performing the different test I am going to do in this project, I will need different hardware devices which are the following ones.

- A Linux system with a kernel version less than 4.15
- An Android phone with a version of wpa_supplicant 2.6
- An IOS device.
- A MAC computer with MACOS operating system
- A WifI antenna Alfa network AWUS036AC
- A Windows 10 Computer

4.2 Software devices

Also I will need different software resources like:

- KRACK scripts developed by M. Vanhoef.
- Aircrack suite
- Linux OS
- Wireshark tool for capturing the handshake
- WifiConn for getting the radio frequency packages in a windows system

5 Keys and key distribution

5.1 EAPOL frames

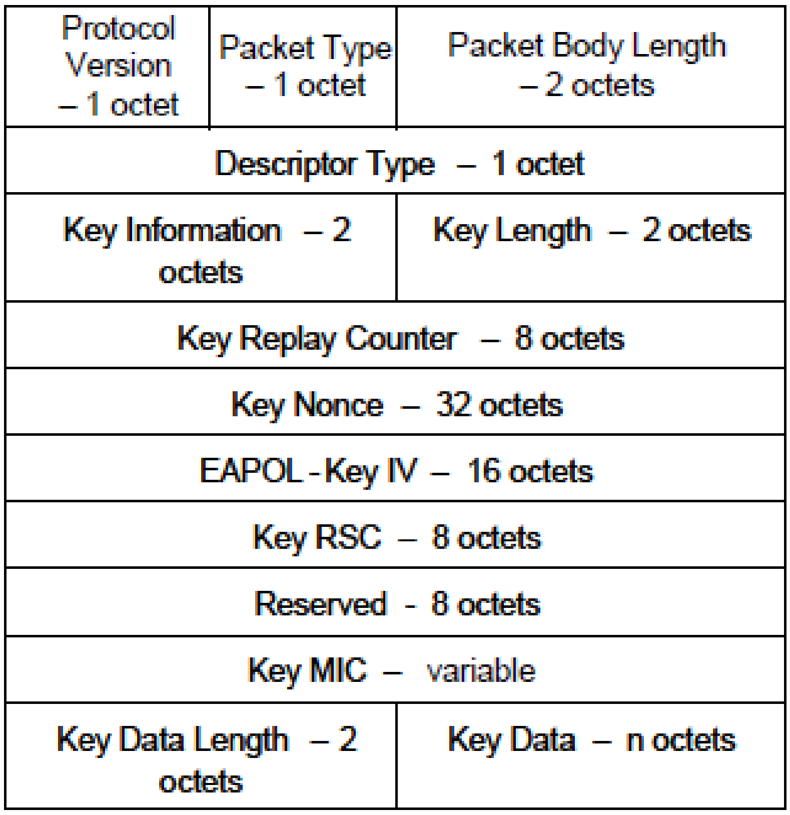
The EAPOL frames are the most important part of this thesis. This is due to these are the frames which are exchange between the stations(STA) when sending information. Because of this exchange the STA can share cryptographic keys which are going to be used in the future to encrypt the information. The three exchanges implemented using EAPOL frames are the following:

- 4-way handshake to confirm the PMK between the STAs is the same and to transfer the GTK to the STA
- Group key handshake to update the GTK at the STA
- Deliver SMK and STK.

Now it is important to define the different types of keys which were mentioned before.

- The PMK is the derived key from the Extensible authentication Protocol (EAP) or directly obtained from a preshared key(PSK)
- The GTK is the Group temporary key. KRACK attack also attacks this Key but for our purposes is not important, because we are going to study the PMK
- The SMK is the station-to-station link master key which is generated by the access point during the SMK handshake and will be used for deriving an STSL trasient key (STK)

The fields conventions for EAPOL-Key frames are defined by the IEE802.1x 2010 and are the following ones:



Protocol Version – 1 octet	Packet Type – 1 octet	Packet Body Length – 2 octets
Descriptor Type – 1 octet		
Key Information – 2 octets		Key Length – 2 octets
Key Replay Counter – 8 octets		
Key Nonce – 32 octets		
EAPOL-Key IV – 16 octets		
Key RSC – 8 octets		
Reserved - 8 octets		
Key MIC – variable		
Key Data Length – 2 octets		Key Data – n octets

Figure 5.1: EAPOL frame from the 802.11 standard (2016 version)

- The descriptor type has a value defined by IEEE 802.1x
- Key information has two octets which specifies the characteristics of the key.
  - The first two bits are the Key description version which must be set to 0 on all trasmited EAPOL-KEY frames except in a defined situations:
    - If the value is 1 it indicates that
      - HMAC-MD5 is used in the Key MIC[7]
      - The algorithm used for the key data field is ARC4
      - The MIC is 16 octets
    - If the value is 2:
      - HMAC-SHA-1-128 is used in the Key MIC[8]
      - The algorithm used for the key data field is NIST AES
      - The MIC is 16 octets

  - The next bit (bit 3) are key type is used to specifies if the EAPOL-Key frame is part of the 4-way handshake deriving a PTK. The value 0 indicates that it is not part of a PTK derivation and 1 means that it is part of a PTK derivation
  - Bits 4 and 5 reserved
  - The next bit is the installation one (bit 6) and it is one the temporal key derived from the STA must be configured. If it is 0 means that it shall not be configured. This bit is only used if the (bit 3) is 1 if not it is reserved
  - The next bit (bit 7) is the ACK bit which is used for notice that it must be an answer for that EAPOL-Frame
  - Key MIC (bit 8) is set to 1 if there is a MIC and to 0 if it is not.
  - The secure bit (bit 9) is set to 1 when the initial Key exchange is finished.
  - The bit 10 is the error bit
  - The request bit (bit 11) it is used to request the initialization of the 4-way handshake or the group key handshake
  - The encrypted data bit is set to one if the Key data field is encrypted
  - The SMK message (bit 13) specifies when the EAPOL-frame is part of the SMK handshake
  - The bits 14-15 are reserved

- Key length: It is 2 octets length and defines the length of the PTK.
- Key Replay counter: 8 octets which are initialized to 0 when the PMK is initialized.
- Key nonce: The field is 32 octets. It conveys the ANonce and the SNonce
- EAPOL-Key IV: This field is 16 octets. It shall contains a 0 if the IV is not required.
- Key RSC: It contains the receive sequence counter (RSC) for the GTK being installed. It is used in the message 3 of the 4-way handshake.
- Key MIC from and including the EAPOL protocol version field to and including the Key Data field, calculated with the Key MIC field set to 0
- Key Data Length. This field is 2 octets in length and represents the length of the Key Data field in octets.
- Key data: This field is which is compose of any additional data which is needed for the key exchange and it is not in the other EAPOL-Frame fields.

5.2 4-way handshake

Now I am going to explain which are the EAPOL-Frames which are stated by the standard while performing the 4-way handshake. Because it is a 4-way handshake there will be 4 different frames which are going to be sent from one STA to the other.

The message 1 is sent by the authenticator to the supplicant. In this message there are three possibilities for the key data field. To be 0, to be the PMK-ID for generating an specific PTK or PMK-ID KDE for STK generation

The message 2 is the answer to the message 1, this means it must have the same Key Replay Counter than the message 1, and in the Key data field it will have the RSNE (Robust secure network element) for creating the PTK.

The message 3 sends to the supplicant the GTK and the RSNE for creating the PTK

The message 4 is an ACK for the message 3 and let know the authenticator that the supplicant has installed the key successfully.

It is important to know how the Key replay counter works in each message, because the KRACK attack is based in this field to perform the attack. So, in the first message the Key Replay Counter is set to a number (n), which in most of the cases is 0. The answer provided by the supplicant, message 2, will have the same Key Replay counter. Then the authenticator will send the message 3 with the Key Replay Counter n+1 and the answer must have the same Key Replay counter (n+1). The summary of this message exchange is that the supplicant can't modify the Key Replay Counter field and only the authenticator will increment it.

There are some restrictions with the Key Replay Counter field value, its value must be more than to the current local value, if it is equal or less the EAPOL-frame will be discarded. If the Key Replay counter field has a valid value, the supplicant will generate its nonce (SNonce), it will derivate the PTK and construct the message 2 with this information and the one received in message 1.

When the message 2 is received by the authenticator it will check the Key Replay counter, and if it is correct, if it is not correct it will silently discard the message. In this transaction an in the rest of the messages the MIC field will be checked for integrity purpouses. Now if the supplicant is performing an association it should send an old RSNE value to use a already installed Key. If the RSNE value is an old one the handshake will be finished here, but if it is not, the authenticator will construct the message 3.

This message 3 has the ANonce, because is a message sent by the authenticator to the supplicant and in the Key Data field it will have the information for the PTK generation, the RSNE for the current operating band and optionally it will have the information related to the GTK if this is been negotiated. In this message is in the one the KRACK attack is going to exploit the vulnerability.

When receiving this message the supplicant will not only check the Key Replay Counter, it also would check the ANonce and the MIC field, so it can analyze if the message has been modified or if an unauthorized entity has send the message. This countermeasures do not work against replay attacks, and because of that KRACK is based in this type of attacks.

The steps done by the supplicant when receiving the message 3 and after checking the Key Replay Counter and the ANonce will be the following ones:

- Verifie the RSNE, PMKID or the PKM1Name must be identical to the one sent in the message 1.
- Verifies the MIC of message 3, if doesn't match with the previous received, the supplicant will discard silently the message 3
- Generates the message 4 and send it to the Authenticator

6 KRACK exploit

The KRACK attack was developed by Mathy Vanhoef to exploit a vulnerability found in the cryptographic protocols to install a already-in-use-key in the supplicant. This is achieved by reinstalling information like the nonces and the replay counter. Depending of what encryption protocol is being used in the 4-way handshake the impact is more or less severe. For example if the encryption protocol which is been used is AES-CCMP the unauthorized user can reply and decrypt packages although she can't forge them. The worst case is against TKIP and GGMP[9] because with this protocols the unauthorized entity will be able to reply, decrypt and forge.

Before the KRACK attack the only vulnerability which had been found was in TKIP[10]. TKIP was only developed as a short time solution during CCMP was being developed. CCMP was proven as secure in different papers.
This attack is absolutely devastate against the versions 2.4 and 2.5 of the wpa_supplicant which is usually used in Linux systems. Because Android is a Linux kernel modification in the version 6.0 of the Android System and in the version 2.0 of the Android Wear this vulnerability is also found.

In the last section I have explained deeply different cryptographic concepts which are applied during the 4-way handshake but now I am going to write a summary of the more important parts which are involved in this attack performed:

The mutual authentication is based on a Pairwise Master Key which is shared by a symmetric encryption protocol and it is used for derivating the Pairwise Temporal Key (PTK). This PTK is not only obtained using the PMK it also uses the MAC addresses and the nonces.

This PTK is split into a Key Confirmation Key (KCK), a Key Encryption Key (KEK) and a Temporal Key (TK). The KCK and KEK are used to protect handshake messages, while the TK is used to protect normal data frames with a data-confidentiality protocol. The KRACK paper states that the authenticator must increment the key replay counter each time it is replaying a frame and the supplicant will use the same Key Replay counter as the frame he has received before. This briefly explains what I have mentioned before about the Key Replay Counter but in the next section where I am going to state the hypothesis I have followed during this research I will explained more deeply why this point is the start of the our hypothesis.

About how the handshake works it is perfectly explained in the KRACK paper using a state machine with the different paths which are followed depending which the situation is.

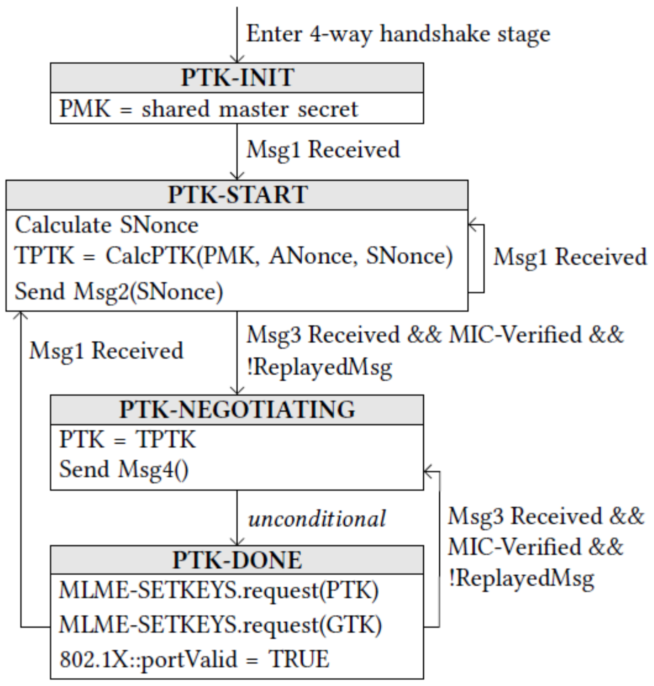

Enter 4-way handshake stage
PTK-INIT
PMK = shared master secret
Msg1 Received
PTK-START
Calculate SNonce
TPTK = CalcPTK(PMK, ANonce, SNonce)
Send Msg2(SNonce)
Msg1 Received
Msg1 Received
Msg3 Received && MIC-Verified && !ReplayedMsg
PTK-NEGOTIATING
PTK = TPTK
Send Msg4()
unconditional
Msg3 Received && MIC-Verified && !ReplayedMsg
PTK-DONE
MLME-SETKEYS.request(PTK)
MLME-SETKEYS.request(GTK)
802.1X::portValid = TRUE

Figure 6.1: Supplicant 4-way handshake state machine as defined in the 802.11 standard and mentioned in the KRACK paper)

6.1 Key Reinstallation attack

The attack is based in the idea that the supplicant can still receive retransmisions of the message 3 although if it is in the PTK_DONE state show in the machine state diagram above. So the idea is to perform a MiMT attack between the supplicant and the authenticator so it prevents the message 4 from arriving to the authenticator. This will result in a retransmision of message 3, which causes the supplicant to reinstall an already-in-use PTK. The problem is that Windows and IOS systems are not vulnerable because they dont implement correctly the state machine. Also the MiMT attack has another difficulty because you can't only set up a rouge AP with a different MAC address and then forward packages because the session key is based in the MAC address. So a channel-based MiMT attack must be performed.

When the MiMT is performed the attacker will forward the messages 1 and 2 as if the attack was not happening. When she receives the message 3, she will forward it to the supplicant and she won't forward the message 4 received as an answer to that message 3. In this moment the supplicant will have installed the PTK and will start sending frames, this is because the message 4 is only an ACK for the message 3 and to let the authenticato know that the supplicant is going to install the PTK. However, because the authenticator hasn't receive the ACK (message 4), he will resend the message 3. The attacker in this moment will forward the new message 3 to the supplicant.

As the protocol states the supplicant will answer to this message 3 with a message 4 as an ACK. However, this message will be encrypted using the previous PTK installed. This message will arrive to the unauthorized user who will forward it to the authenticator, but also, he will replay the previous message 4 in plain text.

Because the authenticator hasn't installed the previous PTK, because he hasn't received the message 4 of ACK to the previous message 3, the authenticator will discard the encrypted message 4 and accept the one which is in plaintext. The team which has developed this attack uses the following diagram to show how the attack is performed.

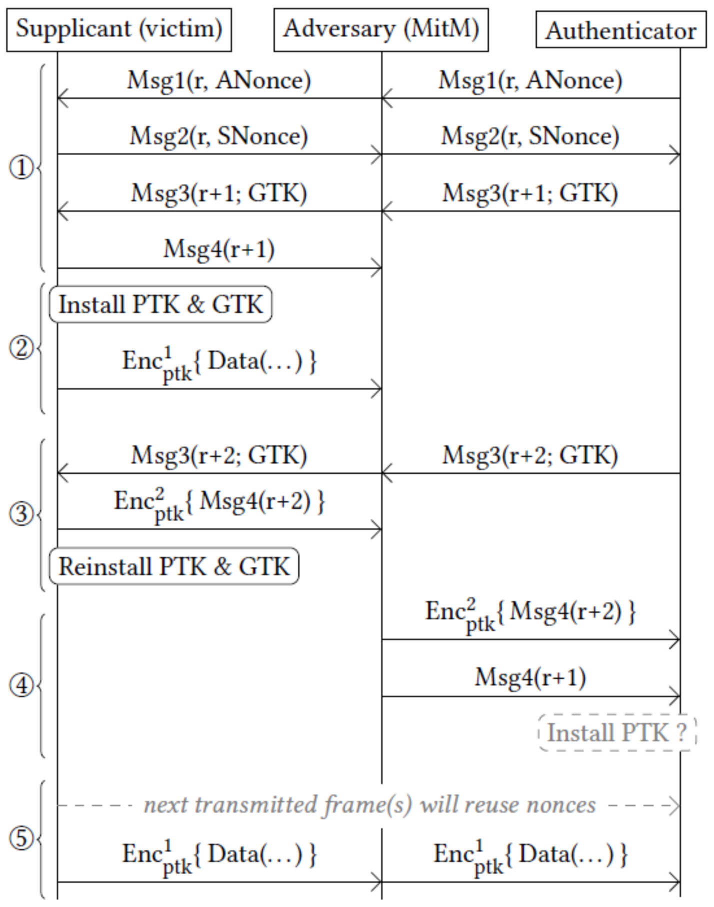


Figure 6.2: Key reinstallation attack against the 4-way handshake defined in the KRACK paper

This attack as it has been explained only can be done if the supplicant still accepts retransmision of the message 3 in plain text after installing the PTK and the GTK. This for example happens with the Android systems which allows plain text retransmisions of message 3 immediately after the supplicant has sent the message 4.

There are other systems which doesn't allow plain text retransmisions of the message 3. The KRACK exploit can be also used in this situations, the way in which is performed is by performing two 4-way handshake.In the first one the unauthorized entity allows the 4-way handshake to be performed properly and the PTK and GTK to be installed. After this the attacker will disassociate the supplicant so he needs to perform another 4-way handshake. This will be encrypted with the PTK which was installed in the first 4-way handshake, so when the first message 3 arrives the attacker won't retransmit it immediately, he will wait until he receives the second message 3 and then send the two messages 3 to the supplicant. Because there are encrypted using the first PTK the supplicant will accept them and answer to the two of them, so the attack can be performed.

An example of a system which only accept retransmited messages is the MAC system or the OpenBSD

This type of attack can be seen in the following diagram in which only the attacker and the supplicant are represented, this is because the part of the authenticator works in the same way as in the previous attack.

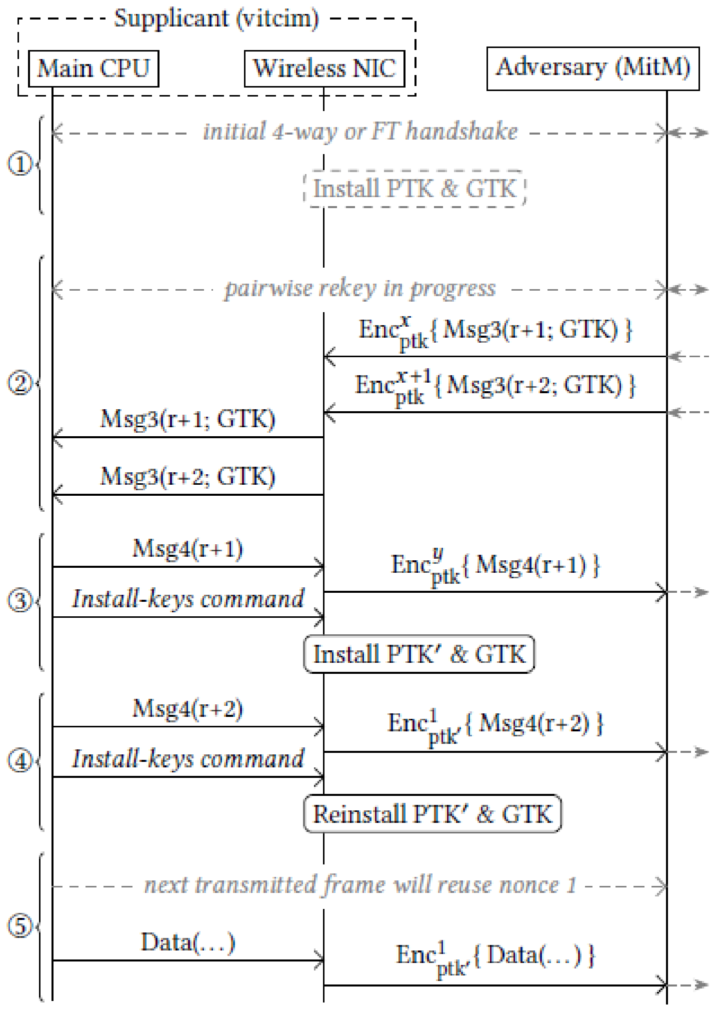

Supplicant (vitcim)
Main CPU
Wireless NIC
Adversary (MitM)
initial 4-way or FT handshake
Install PTK & GTK
pairwise rekey in progress
Enc^{x}_{ptk} { Msg3(r+1; GTK) }
Enc^{x+1}_{ptk} { Msg3(r+2; GTK) }
Msg3(r+1; GTK)
Msg3(r+2; GTK)
Msg4(r+1)
Enc^{y}_{ptk} { Msg4(r+1) }
Install-keys command
Install PTK′ & GTK
Msg4(r+2)
$Enc^{1}_{ptk'}$ { Msg4(r+2) }
Install-keys command
Reinstall PTK′ & GTK
next transmitted frame will reuse nonce 1
Data(...)
$Enc^{1}_{ptk'}$ { Data(...) }
①
②
③
④
⑤

Figure 6.3: Key reinstallation attack against the 4-way handshake, when the victim only accepts encrypted message 3

After this explanation and looking for the diagram above we can find the question to the first hypothesis I am going to investigate, which are the following:

- Does the authenticator have to accept the plain text message 4 because it is using an old Key Replay Counter?
- Must the Key Replay counter be incremented although the EAPOL frame sen is a retransmision?

7 Hypothesis 1

7.1 Hypothesis description

After explaining how the 4-way handshake work I started investigating about our first hypothesis. The hypothesis was stated by Mr. David Rose Sr. Network Engineer at Illinois Institute of Technology. The hypothesis says the following statement: **"There could be a misunderstood in the analysis of the standard 802.11 when affirming that the implementation which has been done by Windows and IOS is incorrect because the 802.11 standard states: On reception of message 3, the STA_P silently discards the message if the Key Replay Counter field value has already been used"** .

This makes sense because in other protocols of the OSI model, like TCP, the replayed messages has the same sequence number than the original one and if this is the situation the supplicant should discard silently this message and the attack will not be performed, so the ones which would have implemented correctly the protocol would be IOS and Windows. For that we are going to first see how the attack works on a IOS system and in a Linux one, so we can see how the attack was developed and then analyze deeply the standard looking for more information about how this reply messages are performed.

In the different references which can be found in this thesis different parts related to the security protocols or the 802.11 standard are properly explained. However, there is no information about the message 3 or the message 4 of the handshake.

7.2 KRACK attack proof of concept

The KRACK attack code is private and only the KRACK paper group has the code itself, but they have uploaded to github some scripts which allows to check if one system is vulnerable to this attack or not. This is what I am going to use to check if the Linux/Android systems are vulnerable and if the IOS and Windows system arent́ as the KRACK paper state.

7.2.1 Installation of the KRACK tools

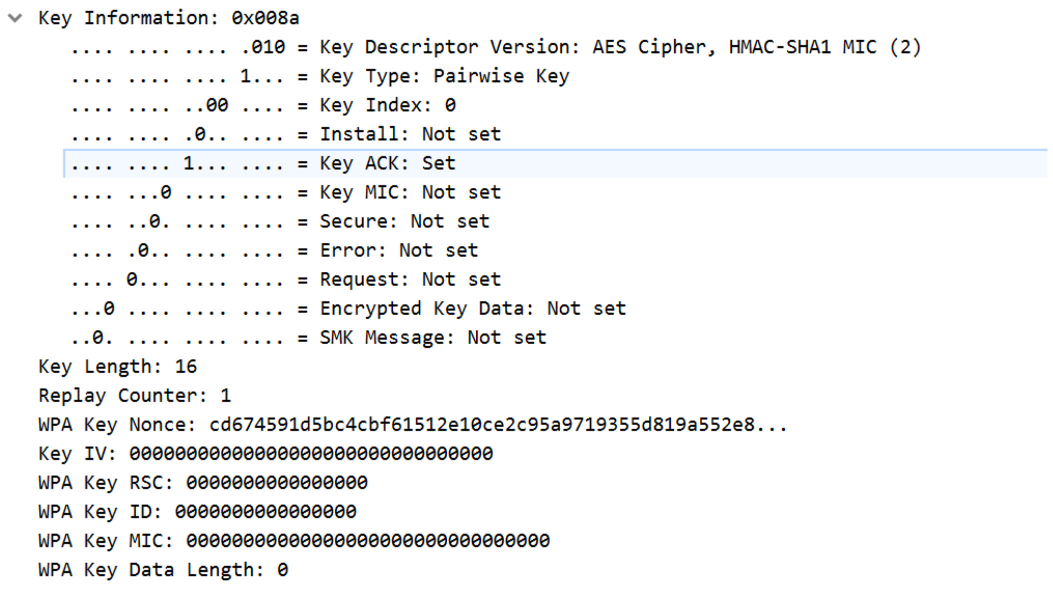
This script were developed for a Kali Linux system and when they are used in other type of systems like an Ubuntu 18.04 the Wifi driver has some problem when trying to run the hostapd of the attack. The problem is caused by the NL80211 which is the 802.11 net link which debian uses for their wireless driver. After installing the Kali Linux system I will first disable the hardware encryption, the hardware encryption could interfere with this script. Also, I have used the airmon suite to check that the network manager is off and that there is not other hostapd process running.
When I had disabled the hardware encryption I am going to compile the hosapd file using the regular unix command **make**.

There are some problems that could happen during the installation of the KRACK scripts. *One of the worst problem which could happened is that the computer which the tester is using is so old that the Wireless card could not act like an AP, so he wouldn't be able to perform this attack. The only possible solution for this situation is buying, installing and using an external wireless antenna which can be used as an AP*

When I was doing the attack I was capturtig with Wireshark the different messages from the handshake which were sent between the supplicant and the authenticator. This will help me to analyze how the attack works and what is the value of the Key Replay Counter received in each package.

7.2.2 4-way handshake messages captured

I am going to analyze each package against a vulnerable device. For starting the handshake the authenticator send the message 1:



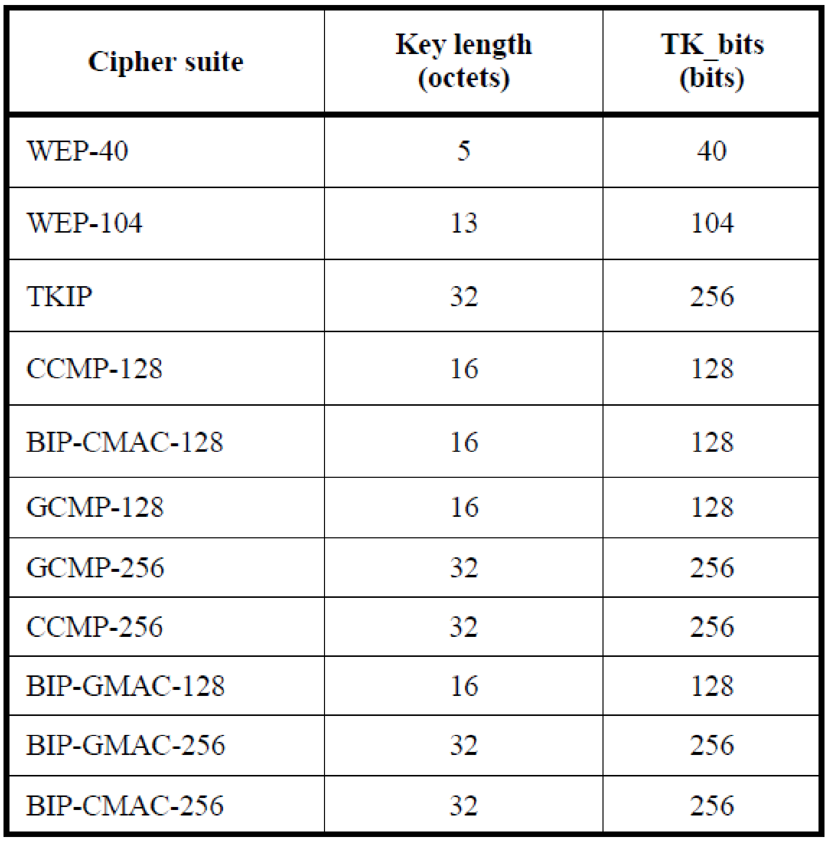
```
Key Information: 0x008a
    .... .... .... .010 = Key Descriptor Version: AES Cipher, HMAC-SHA1 MIC (2)
    .... .... .... 1... = Key Type: Pairwise Key
    .... .... ..00 .... = Key Index: 0
    .... .... .0.. .... = Install: Not set
    .... .... 1... .... = Key ACK: Set
    .... ...0 .... .... = Key MIC: Not set
    .... ..0. .... .... = Secure: Not set
    .... .0.. .... .... = Error: Not set
    .... 0... .... .... = Request: Not set
    ...0 .... .... .... = Encrypted Key Data: Not set
    ..0. .... .... .... = SMK Message: Not set
Key Length: 16
Replay Counter: 1
WPA Key Nonce: cd674591d5bc4cbf61512e10ce2c95a9719355d819a552e8...
Key IV: 00000000000000000000000000000000
WPA Key RSC: 0000000000000000
WPA Key ID: 0000000000000000
WPA Key MIC: 00000000000000000000000000000000
WPA Key Data Length: 0
```

Figure 7.1: Message 1 of the 4-way handshake

In this first message we can see the most important fields of the EAPOL frame which were explained in the EAPOL frames 5.1 analysis done before. First we can see that the cypher which is being use is the AES cypher with a MIC value obtained by a HMAC-SHA1 algorithm.

As I have explained before if the AES crypto algorithm is being used during the transmision of the data, the unauthorized entity only will be able to decrypt but not forge the packages exchanged between the authenticator and the supplicant.

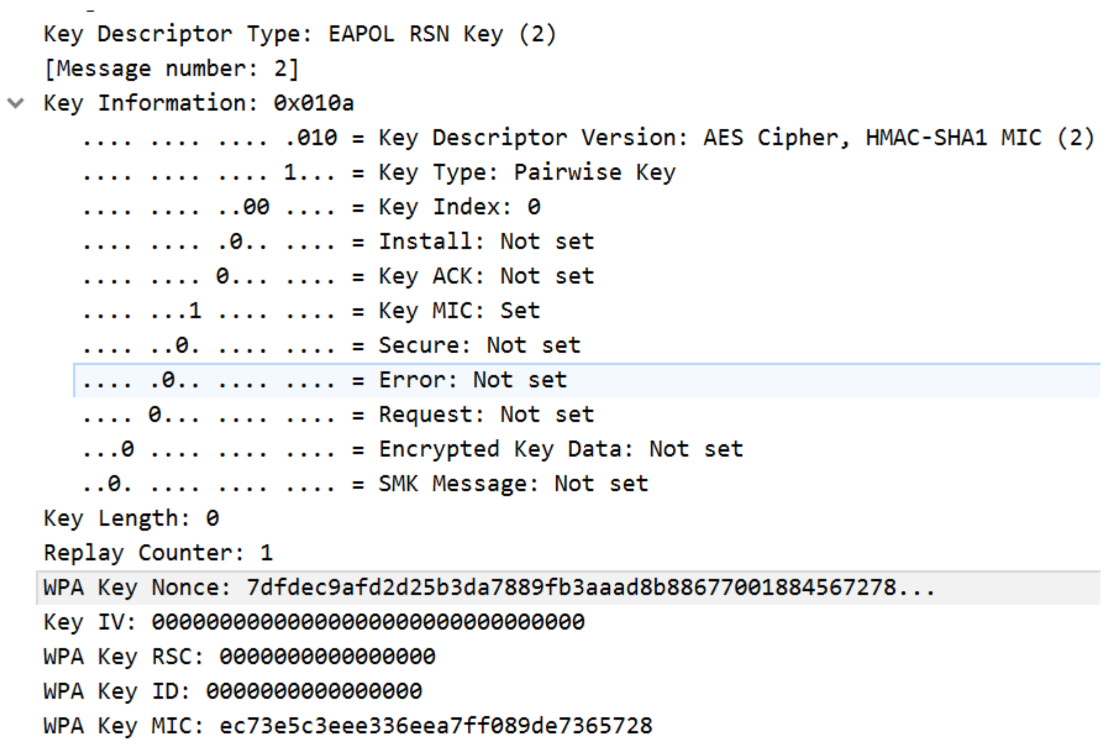
For checking what the key length means we can check the information about the Key length value in the 802.11 standard



Cipher suite	Key length (octets)	TK_bits (bits)
WEP-40	5	40
WEP-104	13	104
TKIP	32	256
CCMP-128	16	128
BIP-CMAC-128	16	128
GCMP-128	16	128
GCMP-256	32	256
CCMP-256	32	256
BIP-GMAC-128	16	128
BIP-GMAC-256	32	256
BIP-CMAC-256	32	256

Figure 7.2: Cypher suite key lengths

So because the value obtained in the message 7.1 is 16 we know that key length will be of 128 bits. The next field is the Key Replay Counter, this is the most important field for our hypothesis. In this first message the value of the Key Replay Counter is 1. This is the regular value for this field in the message 1 when a handshake has just started and is the first message which has been exchanged between the two nodes. Also we can see that we have a value in the WPA Key Nonce which is the nonce related to that entity in the communication. The next image represents the message 2 of the handshake:



```
Key Descriptor Type: EAPOL RSN Key (2)
[Message number: 2]
Key Information: 0x010a
   .... .... .... .010 = Key Descriptor Version: AES Cipher, HMAC-SHA1 MIC (2)
   .... .... .... 1... = Key Type: Pairwise Key
   .... .... ..00 .... = Key Index: 0
   .... .... .0.. .... = Install: Not set
   .... .... 0... .... = Key ACK: Not set
   .... ...1 .... .... = Key MIC: Set
   .... ..0. .... .... = Secure: Not set
   .... .0.. .... .... = Error: Not set
   .... 0... .... .... = Request: Not set
   ...0 .... .... .... = Encrypted Key Data: Not set
   ..0. .... .... .... = SMK Message: Not set
Key Length: 0
Replay Counter: 1
WPA Key Nonce: 7dfdec9afd2d25b3da7889fb3aaad8b88677001884567278...
Key IV: 00000000000000000000000000000000
WPA Key RSC: 0000000000000000
WPA Key ID: 0000000000000000
WPA Key MIC: ec73e5c3eee336eea7ff089de7365728
```

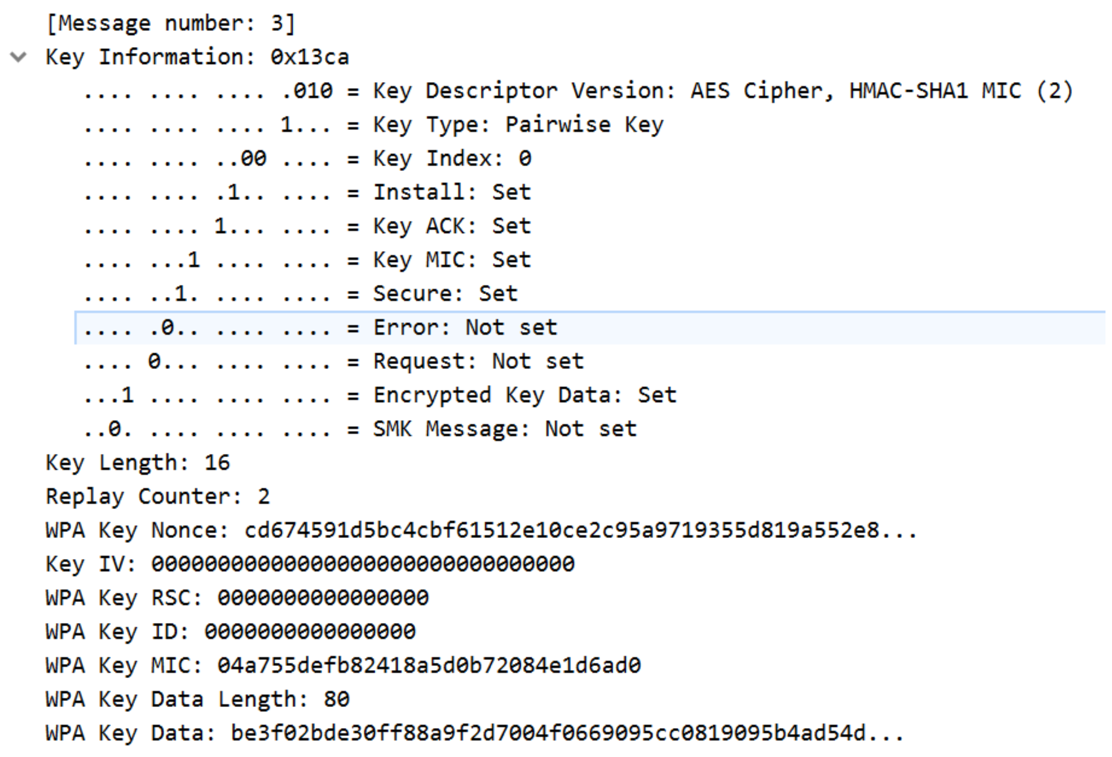
Figure 7.3: Message2 of the 4-way handshake

The message 2 is different from the message 1 in two fields:

- The first one is the Key Information Data in which the ACK bit is not set in this message because it is actually an answer to the previous message7.1 which has the ACK bit set.
- The second one is that the WPA Key nonce value is different from the previous one. This is related to what it was explained before in 5.2. There are two nonces the ANonce and SNonce, this because the message was sent by the supplicant is the Supplicant-Nonce(SNonce) and the one in the 7.1 is the Authenticator nonce (ANonce)

The Key Replay Counter is 1 as in the previous message because the supplicant can't update this value and must always answer to each message with the previous Key Replay Counter obtained.

The next figure shows the message 3 of the handshake:



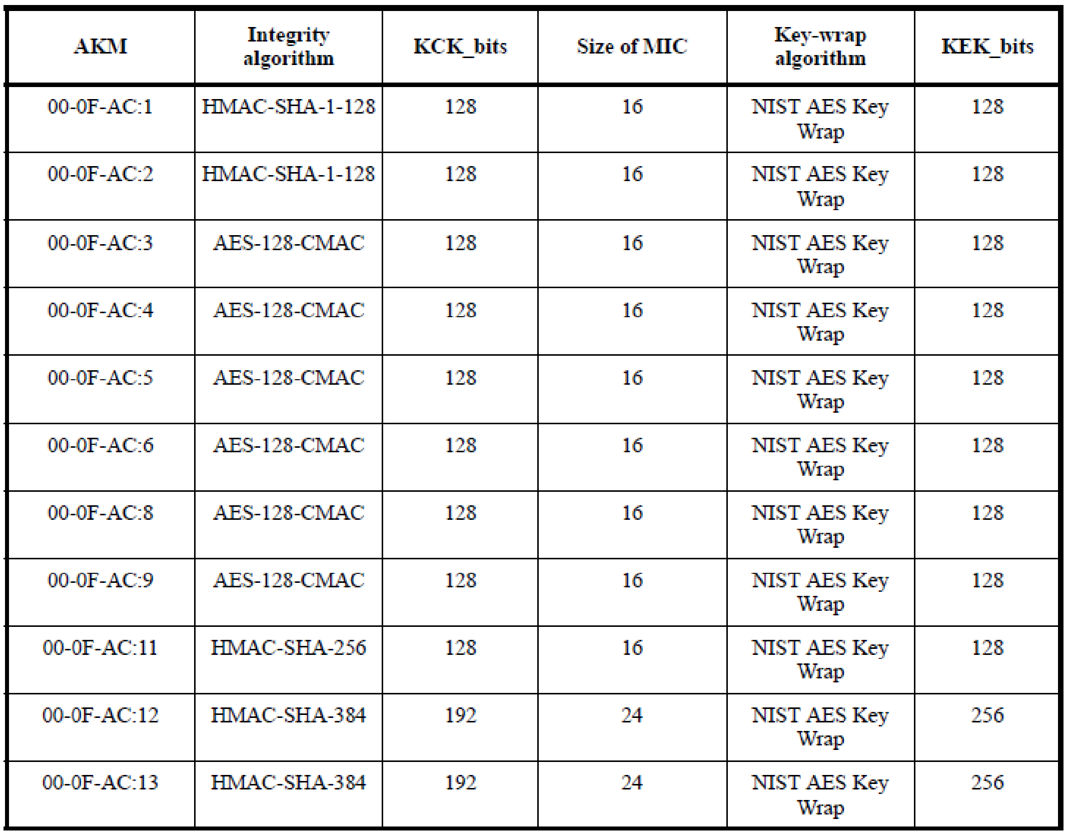
```
  [Message number: 3]
v Key Information: 0x13ca
     .... .... .... .010 = Key Descriptor Version: AES Cipher, HMAC-SHA1 MIC (2)
     .... .... .... 1... = Key Type: Pairwise Key
     .... .... ..00 .... = Key Index: 0
     .... .... .1.. .... = Install: Set
     .... .... 1... .... = Key ACK: Set
     .... ...1 .... .... = Key MIC: Set
     .... ..1. .... .... = Secure: Set
     .... .0.. .... .... = Error: Not set
     .... 0... .... .... = Request: Not set
     ...1 .... .... .... = Encrypted Key Data: Set
     ..0. .... .... .... = SMK Message: Not set
  Key Length: 16
  Replay Counter: 2
  WPA Key Nonce: cd674591d5bc4cbf61512e10ce2c95a9719355d819a552e8...
  Key IV: 00000000000000000000000000000000
  WPA Key RSC: 0000000000000000
  WPA Key ID: 0000000000000000
  WPA Key MIC: 04a755defb82418a5d0b72084e1d6ad0
  WPA Key Data Length: 80
  WPA Key Data: be3f02bde30ff88a9f2d7004f0669095cc0819095b4ad54d...
```

Figure 7.4: Message3 of the 4-way handshake

Here we can see that the Key Information bits are similar to the ones which we can find in the message 1. The only difference is that the Key Data is set to 1, this means that the Key Data field is encrypted as we explained before in 5.1. This information encrypted is the RSNE and the GTK information for generating the PTK.

Also now the field has the MIC value which the Authenticator has created by using the KCK. The KCK is the first part of the PTK key, which when it is derived from the PMK is formed by two different parts, the KCK and the KEK. The length of the KCK depends of the algorithm is going to be used for creating the MIC field. In this situation we can see that the MIC value is calculated using the HMAC-SHA1 MIC algorithm and for checking the length in bits of the KCK we can check the table wich can be found in the 12.7.3 of the 802.11 standard subsection EAPOL-FRAMES creation, which is the following one:

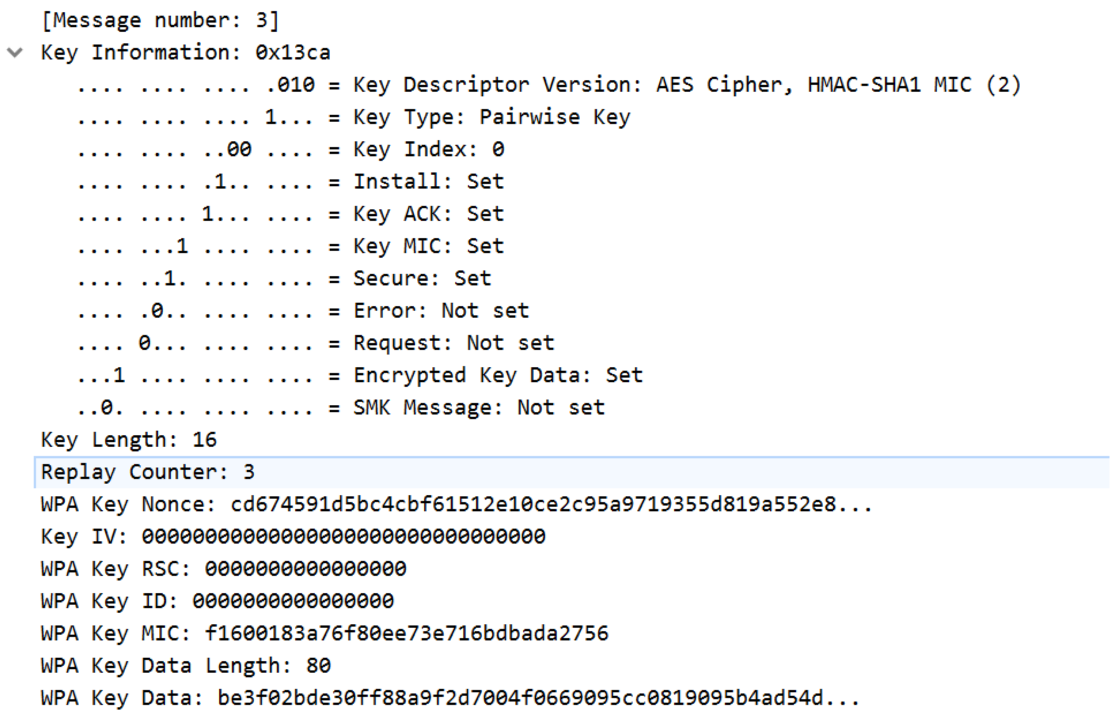
bits.png bits.bb



AKM	Integrity algorithm	KCK_bits	Size of MIC	Key-wrap algorithm	KEK_bits
00-0F-AC:1	HMAC-SHA-1-128	128	16	NIST AES Key Wrap	128
00-0F-AC:2	HMAC-SHA-1-128	128	16	NIST AES Key Wrap	128
00-0F-AC:3	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:4	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:5	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:6	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:8	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:9	AES-128-CMAC	128	16	NIST AES Key Wrap	128
00-0F-AC:11	HMAC-SHA-256	128	16	NIST AES Key Wrap	128
00-0F-AC:12	HMAC-SHA-384	192	24	NIST AES Key Wrap	256
00-0F-AC:13	HMAC-SHA-384	192	24	NIST AES Key Wrap	256

Figure 7.5: Integrity algorithms table

The next message we should found in the authenticator should be the message 4 received from the supplicant as an ACK of the message 3 which has been sent before. However, because this 4-way handshake is compromised, we can see that because the message 4 hasn't arrived to the authenticator, he will send another message 3 and the Key replay Counter is incremented. This is performed by the KRACK attack when re-sending the message 3 and we will later examine the standard to analyze if this is the correct implementation of it.

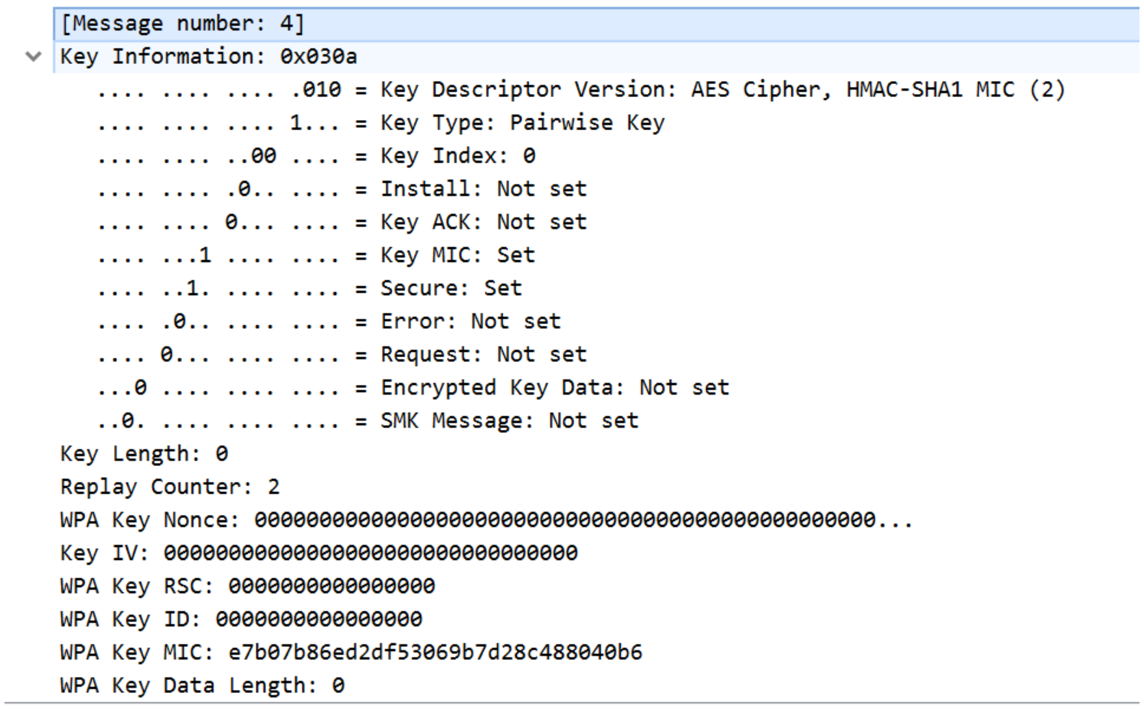


```
  [Message number: 3]
v Key Information: 0x13ca
     .... .... .... .010 = Key Descriptor Version: AES Cipher, HMAC-SHA1 MIC (2)
     .... .... .... 1... = Key Type: Pairwise Key
     .... .... ..00 .... = Key Index: 0
     .... .... .1.. .... = Install: Set
     .... .... 1... .... = Key ACK: Set
     .... ...1 .... .... = Key MIC: Set
     .... ..1. .... .... = Secure: Set
     .... .0.. .... .... = Error: Not set
     .... 0... .... .... = Request: Not set
     ...1 .... .... .... = Encrypted Key Data: Set
     ..0. .... .... .... = SMK Message: Not set
  Key Length: 16
  Replay Counter: 3
  WPA Key Nonce: cd674591d5bc4cbf61512e10ce2c95a9719355d819a552e8...
  Key IV: 00000000000000000000000000000000
  WPA Key RSC: 0000000000000000
  WPA Key ID: 0000000000000000
  WPA Key MIC: f1600183a76f80ee73e716bdbada2756
  WPA Key Data Length: 80
  WPA Key Data: be3f02bde30ff88a9f2d7004f0669095cc0819095b4ad54d...
```

Figure 7.6: Second message 3 sent by the authenticator

We can see that all the fields have exactly the same value except from the Key Replay Counter, which now has the value of 3, and the MIC value which it is logic because the package is not the same as the previous message 3.

After this we can see that the authenticator receives the message 4 related to the first message 3 which has been forwarded by the unauthorized entity.



```
[Message number: 4]
Key Information: 0x030a
    .... .... .... .010 = Key Descriptor Version: AES Cipher, HMAC-SHA1 MIC (2)
    .... .... .... 1... = Key Type: Pairwise Key
    .... .... ..00 .... = Key Index: 0
    .... .... .0.. .... = Install: Not set
    .... .... 0... .... = Key ACK: Not set
    .... ...1 .... .... = Key MIC: Set
    .... ..1. .... .... = Secure: Set
    .... .0.. .... .... = Error: Not set
    .... 0... .... .... = Request: Not set
    ...0 .... .... .... = Encrypted Key Data: Not set
    ..0. .... .... .... = SMK Message: Not set
Key Length: 0
Replay Counter: 2
WPA Key Nonce: 000000000000000000000000000000000000000000000000...
Key IV: 00000000000000000000000000000000
WPA Key RSC: 0000000000000000
WPA Key ID: 0000000000000000
WPA Key MIC: e7b07b86ed2df53069b7d28c488040b6
WPA Key Data Length: 0
```

Figure 7.7: Message 4 sent by the authenticator

The reason to send this old message 4 instead of sending the new message 4 related to the second message 3 is due to when the supplicant sent the first message 4, the one the attacker is not going to forward, it will install the PTK and the GTK so when he receives the new message 3 the message 4 sent as an ACK for this new message 3 will be encrypted with this PTK. The problem is that although the supplicant has installed the PTK and the GTK before sending the message 4 the authenticator will not install it until he receives the ACK. This means that if the attacker replies the *second message 4*, this will be encrypted with a key which the authenticator hasn't installed yet and he will discard this message. This is why the attacker reply with the *first message 4* which is in plain text.

7.3 Analysis of the protocol implementation done by the vulnerable devices

Out hypothesis was based on the following state *On reception of message 3, the STA_P silently discards the message if the Key Replay Counter field value has already been used* and in the idea that a replied message should have the same Key Replay Counter as the original message as other session oriented protocols like TCP do. Also, we wanted to investigate why the message 4 with a a value which is lower than the current local Key Replay Counter value can be accepted by the authenticator instead of being discard silently.

For this I will check the standard looking for more information about this statements. The first statement which I find in the standard related to this hypothesis was the following one. This idea about the Key Reply Counter it is denied by a variable defined in the 802.11 2016 standard call the **TimeoutCtr**.

This variable is used for maintaining the count of the retransmisions of an EAPOL-KEY frame, when this variable is incremented the Key Replay Counter shall be incremented also in each transmision of the EAPOL-KEY frame. In the following image we can see the part of the state machine in which the message 3 is sent by the authenticator the one represented in 6.1 as the PTK negotiating. In this one we can see that this two counters (TimeoutCtr and the Key Replay Counter) are incremented in each timeout.

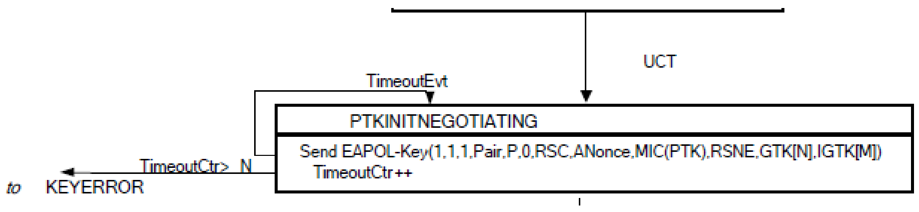

UCT
TimeoutEvt
PTKINITNEGOTIATING
Send EAPOL-Key(1,1,1,Pair,P,0,RSC,ANonce,MIC(PTK),RSNE,GTK[N],IGTK[M])
TimeoutCtr++
TimeoutCtr> N
to KEYERROR

Figure 7.8: PTKINITNEGOTIATING state in the authenticator state machine defined by the standard 802.11 2016

In other references in which other types of attacks are explained like the 4-way Handshake DDOS attack by replying multiples messages 1. In this paper the work group which analyze the possible DDOS attack against the 4-way handshake. In this paper the message 3 is explained deeper than in the 802.11 standard in which the problem about the Key Replay Counter increase in the replayed messages.

This contradicts the first part of our hypothesis, for the second part of it about accepting a message 4 with a sequence number with a value less than the local Key Replay Counter, I am going to analyze the standard looking for information related to replied messages 4. In the 12.7.6.5 part of the 802.11 standard. It states the following: **"On reception of message 4, the Authenticator verifies that the Key Replay Counter field value is one that it used on this 4-way handshake"**. Because the authenticator will accept any message with a Key Replay Counter which has been used in any of the messages which have been sent in the 4-way handshake. This allow the unauthorized entity to perform the attack without any detection of replied messages.

8 Hypothesis 2

8.1 Hypothesis description

After analyzing the first hypothesis, our second hypothesis was about the possible implementation of the WPA2 standard in the Windows and IOS systems. Our hypothesis was that Windows and IOS system don't accept the retransmision of the message 3 because they have implemented the standards without the increase of the Key Replay Counter in the replayed frames.

For analyze this I will modify the code of Krack attack to decrease the Key Replay Counter after resending the message 3. The problem with this idea is that it can't be performed by an unauthorized entity. This is because the the MIC field in the EAPOL-KEY frame is calculated with the value before the increase of the Key Replay Counter. So, when the attacker modifies the EAPOL-KEY frame the MIC value will change and the supplicant will discard the frame because the MIC value recalculated will not be the same as the MIC received from the authenticator.

For testing purposes and using the KRACK scripts I am going to modify the part of the code where the retransmision of the message 3 is formed and decrease the Key Replay Counter for performing the attack without the increase of the Key Replay Counter.

Although I will not be able to perform this attack against the OS because the integrity protocols applied to each EAPOL-Key frame, we want to study if our hypothesis is correct and if Windows and Mac implementation follows our statement although it goes against the 802.11 standard because it doesn't accept replied messages with an incremented Key Replay Counter.

Before developing this theory we analyze the WPA_supplicant code which can be accessed online for checking if there was any information about the implementation which have been done by Apple and Windows in their devices. If we check this information we check that as it is shown the implementation done by Apple and Windows is the same as in the Linux systems which are vulnerable. However, we know that this is not true because the implementation done by Apple in their devices are different, IOS doesn't accept retransmissions and MACOS does.

8.2 Implementation

For that I will modify the config file of the Krack script in which the Key Replay Counter is handled and modify it to perform our test. The results obtained go against this theory because the message 3, is discarded by the supplicant. This means that IOS and Windows devices do not accept retransmisions of the message 3 although they are not following the standard. This implementation could be described as an error but can also be seen as a patch to the vulnerability of this 4-way handshake.

9 Possible solution to the Key Reinstallation attack vulnerability in different OS

After explaining all the vulnerabilities and analyze the different hypothesis about the KRACK exploit and the implementation of the standard in the different operating systems. The main problem which make these OS vulnerable to the Key Reinstallation Attack is due to accepting the message 3 retransmision with or without an injection of a message 1 between the messages 3. The solution I propose to this vulnerability is to modify the standard so it will become more similar to the implementation done by the Windows and IOS systems, in which the message 3 retransmision is not accepted. The idea will be to restart the handshake when the message 4 does not arrive to the authenticator instead of resending the message 3. This will convert the Key Replay Counter to a sequence number which shouldn't be repeated.

Also a modification in the reception of the message 4 should be done for only accepting an EAPOL-KEY frame which has a Key Replay Counter of 4, this will protect the authenticator for retransmisions of EAPOL-KEY frames which has old Key Replay Counter values. With this modifications the 4-way handshake will be protected from the Key Reinstallation Attack and the information will be protected for unauthorized entities.

Also a modification about how the nonces are handled is less important but it should be implemented. If the retransmision of the message 3 is avoided, a nonce reinstallation will be imposible with the KRACK attack. However, if another vulnerability is found, the nonces are the EAPOL-KEY frame fields which should be protected. The nonces are the way in which the attacker is able to get the PTK, so if the nonces are protected for possible reinstallations the protocol Key exchange handshake will be protected from possible attacks. Other possible attacks are DDOS attack by resending messages 1 but this protection is already covered by different security researchers in previous papers like: [11]

10 Conclusions

After the investigation and the study either of the standard, the vulnerability and also the implementation done by each Operating system of the 802.11 standard. There are several conclusions which I was able to extract from this project and which can help any other researcher which wants to study about the 802.11 standar or its implementation in different systems. Also, it is important to know that the WPA3 standard is currently being developed by the IEEE and will have this vulnerabilities patched. However, other types of vulnerabilities will be found in the standard.

The first conclusion of the project is about the KRACK paper and the explotation of the 4-way handshake by resending the message 3 to reinstall a already-in-use key. The first hypothesis was about a flaw in the implementation and not in the standard like the KRACK paper states. However, after the investigation I state that the KRACK paper was right and the problem is with the implementation of the standard and can be solved by modifying the initial handshake.

We decided to study this hypothesis because the implementation of the message 3 in the initial handshake is not clearly defined either in the IEEE standard either in the different papers I have analyzed for developing this project. This could be the reason why Windows and IOS has implemented incorrectly the standard and the reason why they are not vulnerable to the KRACK attack.

The solution I provide in the consequent section helps to prevent possible vulnerabilities in the devices by modifying the standard. However, the KRACK group has developed several steps to secure your system against this attack it will be better to modify the standard and to apply the changes to every system.

Also there is alot of work done and been doing around the 802.11 standard and all the possible attacks which can be performed against this standard. Because of all of this study groups the standard 802.11 is one of the more robust standard in the telecommunications.

11 Appendix

11.1 Code

Listing 11.1: Krack script configuration

```bash
#!/bin/bash


echo "Installing dependencies..."
sudo apt-get update
sudo apt-get install libnl-3-dev libnl-genl-3-dev pkg-config libssl-dev
   net-tools git sysfsutils python-scapy aircrack-ng
sudo apt-get install python-2
sudo pip2 install pycryptodome pycryptodomex
echo "Dependencies installed"
echo "Downloading the scripts"
git clone https://github.com/vanhoefm/krackattacks-scripts
cp krackattacks-scripts/hostapd/deconfig krackattacks-scripts/hostapd/.config
make krackattacks-scripts/hostapd/
echo "Compilation finished"
echo "Disabling the wifi"
sudo ./krackattacks-scripts/krackattack/dissable-hwcrypto.sh
echo "Rebooting the pc for updating the modifications"
sudo reboot
```

```
#!/bin/bash

echo "Stopping network manager..."

sudo service network-manager stop

echo "Killing the processes."

airmon-ng check kill
```

11.2 Problems when using the scripts

When installing the script there are several problems that must be checked before trying to perform the attack. The first one is checking if the wireless card accepts the AP mode, for that the user should use the Airmon-ng suite to check if this is possible. If it is not possible the next option is to connect and external antenna so it can be used as an AP. However, in the last ubuntu version which has the last kernel version, the 4.15, it will not work until the user compiles the drivers by himself. This is because most of the antennas hasn't been craked to work in an AP mode in the last version of the linux systems.

If the user decides to work in kali Linux system another possible problem is about the repositories. The default repositories of Kali Linux are not the regular ones in other Linux systems, so if this drivers needs to be installed the user shall add to the /etc/sources.lst the deb-src line which defines the regular Linux repositories.

Bibliography

[1] William Stallings and Lawrie Brown *Computer Security Principles and practice(2014))*

[2] Mathy Vanhoef and Frank Piessens textitKey Reinstallation Attacks: Forcing Nonce Reuse in WPA2

[3] NIST *Special Publication 800-48 Revision 1 Guide to Securing Legacy IEEE 802.11 Wireless Networks*
`https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-48r1.pdf`

[4] IEEE 802.11b *(1999) https://standards.ieee.org/standard/802_11b-1999.html*

[5] 2013 IEEE 8th International Conference on Industrial and Information Systems *An effective RC4 stream cipher*
`https://ieeexplore.ieee.org/document/6731957`

[6] Ciampa, Mark (2009). *Security Guide To Network Security Fundamentals (3 ed.). Boston, MA: Course Technology. pp. 205, 380, 381.*

[7] RFC 2104 *HMAC: Keyed-Hashing for Message Authentication*
`https://tools.ietf.org/html/rfc2104`

[8] RFC 4634 *US Secure Hash Algorithms (SHA and HMAC-SHA)*
`https://tools.ietf.org/html/rfc8429`

[9] RFC 8429 *Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)*
`https://tools.ietf.org/html/rfc8429`

[10] IEEE 802.11 *2016, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*

[11] MXiaodong Zha and Maode Ma textitSecurity Improvements of IEEE 802.11i 4-way Handshake Scheme

[12] N. Manivannan, P. Neelameham*Alternative pair-wise key exchange protocols (IEEE 802.11i) in Wireless LANs* `International Conference on Wireless and Mobile Communications`

[13] F. D. Rango, D Lentini, and S. Marano,*"Static and dynamic 4-Way handshake solutions to avoid Denial of Service attack in Wi-Fi protected access and IEEE 802.11i"* `EURASIP Journal on Wireless Communications and Networking, vol. 2006, June 2006`

[14] C. He and J. C. Mitchell, *"Analysis of 802.11i 4-way handshake"*, `Proceeding of the ACM workshop on Wireless Security (WiSe '04)`

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

[16] Mathy Vanhoef. *2017. Chromium Bug Tracker: WPA1/2 all-zero session key & key reinstallation attacks.* `(2017). Retrieved August 29, 2017 from https://bugs.chromium.org/p/chromium/issues/detail?id=743276`

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

[18] *Przemysław Machań and Jozef Wozniak. 2013. On the fast BSS transition algorithms in the IEEE 802.11 r local area wireless networks.* `Telecommunication Systems (2013)`

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

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

[21] David McGrew. 2013. IETF Internet Draft: Generation of Deterministic Initialization Vectors (IVs) and Nonces. (2013). Retrieved August 29, 2017 from https://tools.ietf.org/html/draft-mcgrew-iv-gen-03