SHORT DESCRIPTION OF WEP
WITH SPECIAL FOCUS ON RC4

A Thesis
Presented to the
Faculty of
San Diego State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Arts
in
Mathematics

by
YOUSSEF M FRANCIS
Spring 2013
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the
Thesis of YOUSSEF M FRANCIS:

Short Description of WEP
With Special Focus on RC4

Carmelo Interlando, Chair
Department of Mathematics and Statistics

Ricardo Nemirovsky
Department of Mathematics and Statistics

Joseph Lewis
Department of Computer Science

March 27, 2013
Approval Date
Copyright © 2013
by
Youssef M Francis
The wired equivalent privacy is a security standard designed in the year 1999 by the institute of electrical and electronics engineers to provide security in a wireless network. Although WEP was deprecated in 2004 due to many security flaws, it is surprisingly widely used. For instance, in our research, we found an internet site that tracks unique wifi networks in its database claiming that about 25% (of roughly 85356393) of those networks it has documented, still use wired equivalent privacy encryption.

In this thesis, we gave a general descriptive overview of the multiple parts of the security standard. We described the authentication options provided by this standard, along with the error correcting code used in the standard. In this thesis, we also described the types of errors that the error detecting code fails to detect; we then demonstrated how the error detecting code is fooled into accepting certain codewords even though these codewords were not error free. In this thesis, we also described the encryption algorithm “Ron’s Code 4” which is one of the major parts of the security standard since it is used to guarantee the privacy of information. After, describing in details how the encryption and decryption is done, we describe one of the most damaging attacks on the security standard. The Fluhrer-Mantin-Shamir attack is one of the earliest attacks that took advantage of certain initialization weaknesses in the encryption algorithm to give attacker knowledge of the secret key used by the encryption code, thus giving the attacker free access to all the private data encrypted using this key. In this thesis, we described why this attack was so successful, and we demonstrated the effectiveness of the attack with examples.
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>ABSTRACT</th>
<th>iv</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>GLOSSARY</td>
<td>ix</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Contribution of this Work</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Organization of this Work</td>
<td>2</td>
</tr>
<tr>
<td>2 CRYPTOGRAPHY</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Cryptography</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Symmetric-key And Public-key Cryptography</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Stream Ciphers And Block Ciphers</td>
<td>5</td>
</tr>
<tr>
<td>3 WEP</td>
<td>8</td>
</tr>
<tr>
<td>3.1 Definition</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Choosing the IV</td>
<td>10</td>
</tr>
<tr>
<td>3.3 Authentication</td>
<td>11</td>
</tr>
<tr>
<td>3.4 WEP Encryption</td>
<td>14</td>
</tr>
<tr>
<td>3.5 WEP Decryption</td>
<td>15</td>
</tr>
<tr>
<td>3.6 Error Detection</td>
<td>15</td>
</tr>
<tr>
<td>4 RC4</td>
<td>23</td>
</tr>
<tr>
<td>4.1 Some History</td>
<td>23</td>
</tr>
<tr>
<td>4.2 Notation</td>
<td>25</td>
</tr>
<tr>
<td>4.3 Description Of The RC4 Algorithm</td>
<td>26</td>
</tr>
<tr>
<td>4.4 Simple 5-byte Example</td>
<td>28</td>
</tr>
<tr>
<td>5 ATTACKS ON WEP</td>
<td>33</td>
</tr>
<tr>
<td>5.1 Feasibility of attacks on WEP</td>
<td>33</td>
</tr>
<tr>
<td>5.2 List Of Attacks</td>
<td>34</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>CRC of $x^5 + x^4 + x^3 + x$ By Method 1</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>CRC of $x^5 + x^4 + x^3 + x$ By Method 2</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>CRC of $x^7$</td>
<td>21</td>
</tr>
<tr>
<td>3.4</td>
<td>CRC of $x^5 + x^3 + x$</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>RC4</td>
<td>27</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Open key authentication</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>Shared key authentication</td>
<td>12</td>
</tr>
<tr>
<td>3.3</td>
<td>802.11 frame</td>
<td>13</td>
</tr>
<tr>
<td>3.4</td>
<td>WEP encryption</td>
<td>14</td>
</tr>
<tr>
<td>3.5</td>
<td>WEP decryption</td>
<td>16</td>
</tr>
<tr>
<td>4.1</td>
<td>RC4-PRGA</td>
<td>29</td>
</tr>
<tr>
<td>4.2</td>
<td>RC4 KSA</td>
<td>30</td>
</tr>
<tr>
<td>4.3</td>
<td>RC4 PRGA</td>
<td>32</td>
</tr>
</tbody>
</table>
GLOSSARY

CRC  cyclic redundancy check, this is an error detection code. CRC-32 is a 32-bit polynomial commonly used in WEP.

FMS  one of the most devastating attacks on WEP. It is known as the FMS attack after the three cryptographers: Scott Fluhrer, Itsik Mantin, and Adi Shamir who first described the attack.

IEEE  Institute of Electrical and Electronics Engineers. This is an organization best known for developing some standards in the computer and electronics industries.

IV  the initialization vector is a list of numbers, or an input of fixed size, required to start a certain algorithm.

KSA  key-scheduling algorithm. This is the first part of RC4. The purpose of KSA is to initialize the permutation used in the second part of RC4 known as the PRGA.

PRGA  pseudo-random generation algorithm. This is the second part of RC4. The purpose of PRGA is to generate pseudo-random output bytes.

RC4  this is the stream cipher used in WEP. RC4 stands for Ron’s Code 4, after the name of it’s designer Ron Rivest.

TKIP  The temporal key integrity protocol is a security protocol designed to replace WEP without requiring the replacement of hardware.

WEP  wired equivalent privacy is a security algorithm for IEEE 802.11 wireless networks.

WPA, WPA2  Wi-Fi protected access and Wi-Fi protected access 2 are two security protocols developed by the Wi-Fi Alliance to secure wireless computer networks.
ACKNOWLEDGMENTS

I would like to thank Dr. Interlando for being my mentor and allowing me to work on this thesis. For all the time he invested and all the support he provided. I would also like to thank Dr. Nemirovsky and Dr. Lewis for their time and efforts in helping me with this thesis endeavor.

Also, many thanks to my family and especially my parents, and to the rest of my family for their sustained support throughout the many years.
CHAPTER 1
INTRODUCTION

As our society makes new advances in many technological areas, we begin to become more dependent on the Internet to share and distribute the wealth of information that we obtain. Many of us can already observe that the Internet plays a huge role in our daily lives. In addition to that, wireless devices, for example: phones, have become economical and easily produced. Thus, they have become available to most anyone. This means that more and more people are using their wireless devices to connect to the Internet. For some these devices offers an escape from reality, given all the activities that one can participate in; others use wireless devices to conduct business transactions. Many more use the Internet as a way to connect and communicate to others.

Whatever the reason or motivation to use the wireless devices and the Internet, many, if not all, of the users share a common concern. The common concern is how to keep data private, away from inquisitive and malicious parties.

This privacy problem has been an issue since the dawn of man, but it became of major importance with the spread of wireless devices. To deal with this, standards have been put in place to guarantee measures of privacy. WEP was one of the early standards set up to provide users with privacy equivalent to the privacy obtained by using a wired device. Although advances have been made in the field of securing wireless networks, WEP still enjoys some use to this day [22]. However, many users are recommended to move away from the old WEP created in the late 1990s to use WPA2 which provides better security.

1.1 CONTRIBUTION OF THIS WORK

In this work, we have decided to describe WEP focusing on certain aspects of the standard including the encryption schemes employed. We also focus on the attack that rendered WEP insecure. One of our goals is to collect all the necessary information related to WEP and present the findings in a single document. Another one of our goals is to simplify much of the technical details in order to give the reader some knowledge of what happens in the background when wireless connection to a network is made. Yet another one of our goals in working on this thesis is to have a self contained document. In order to accomplish this goal, we have decided to include some mathematical ideas, normally studied in the area of cryptography, so that any reader may understand our work even when the reader does not have a deep understanding of cryptography. We also decided to include some of the vulnerabilities
of WEP and the different ways to attack WEP, not to encourage the reader to take advantage of these weakness, but on the contrary, to consider avoiding such weakness when designing any new system.

### 1.2 Organization of this Work

Our work is organized in the following manner: in Chapter 2, we first start by defining the word “cryptography” and provide some basic examples of cryptographic algorithms along with some definitions.

In Chapter 3, we give an overview of how WEP operates. We provide a description of the different parts required in order for one to be able to use WEP. In particular, in Section 3.6, we discuss the error correcting code used in WEP, and additionally, we describe one of the weaknesses of this error correcting code which lead to the departure away from WEP and to the developments of other standards such as WPA and WPA2.

Chapter 4 is dedicated to describing Ron’s Code 4 (also known as RC4), the cryptographic algorithm used in WEP. For instance, in Section 4.1 we discuss some of codes’ history; in Section 4.4, we describe the inner workings of the code through an example.

In Chapter 5, we will move on to describing some of the shortcomings of WEP, we provide short descriptions of a few different types of attacks that take advantage of certain design weaknesses in WEP.

In Chapter 6, we focus on the Fluhrer-Mantin-Shamir attack which is one of the oldest and most known attacks on WEP. Using the ideas presented in the sections of this chapter, an attacker can obtain valuable and confidential information that allows the attacker to eavesdrop on conversations. This attack is demonstrated with an example in Section 6.5.

Chapter 7 is the conclusion of our work. In this chapter we also describe some of the advances that have been made. These areas of study are related to our subject and the reader can study or do future research on.

In the appendixes attached to this work, we provide some of the C++ code used to generate and check the results obtained in Section 4.4 and again in Section 6.5. This code is provided for the reader so that it can be used in any future studies of WEP.
CHAPTER 2
CRYPTOGRAPHY

In this chapter, we define some of the terms used in cryptography. We also give a short example in which we explain one of the basic algorithms of cryptography. We define symmetric-key and public key cryptography and also mention some of benefits of each. We also describe some of the differences between stream ciphers and block ciphers.

2.1 CRYPTOGRAPHY

In this section we define the word cryptography and describe one of the oldest cryptographic algorithms known as the shift cipher. The word cryptography has its origins from the Greek word kryptos (κρυπτός) which means “hidden, secret.” The role of cryptography is to make communication secure between parties especially in the presence of an adversary.

In his book “Cryptography: Theory and Practice,” Douglas Stinson describes the objective of cryptography as follows:

The fundamental objective of cryptography is to enable two people to communicate over an insecure channel in such a way that an opponent cannot understand what is being said.

In many cryptography books, the two people communicating are usually given the names Alice and Bob, while the opponent is known as Oscar.

Definition 2.1. Plaintext: the information that Alice wishes to send to Bob before encryption.

Note: the information to be sent is completely arbitrary. It can be English text, numerical data, music, pictures... Alice encrypts the information using a method of her choice, to obtain the ciphertext.

Definition 2.2. Ciphertext: the encrypted information that Alice wishes to send to Bob.

Cryptography relies on different techniques and algorithms, or protocols, to keep the plaintext private and comprehensible only to the intended parties. To an outsider who has no knowledge of the secret key, information will be scrambled and will look as random sequences of letters and characters without any special pattern. However, given the right tools, these random sequences of letters and characters can be returned into meaningful information and more importantly, the data is returned to its original form i.e., before encryption.

Other uses of cryptography involve dealing with various aspects in information security such as data confidentiality, data integrity, and authentication.
The reader may already be familiar with the **substitution cipher** in which letters or numbers are substituted with either letters, numbers, or random characters. Many newspapers or puzzle games would then challenge the reader to decrypt different combination of letter means. In this setting, the player is given a chart defining the appropriate substitutions, for instance, the puzzle could define that \( a = 01 \), \( b = 02 \), \( c = 03 \), ..., \( z = 26 \). The goal of the player would then be to decrypt certain messages such as “13 01 22 08” which would decrypt to “math.” Note that a different puzzle could define the alphabet letters differently. This means that the reader will no longer be able to decrypt “13 01 22 08” correctly, unless given a chart describing the new definitions.

The **shift cipher**, also known as the **Caesar cipher**, is one of the oldest ways and most basic ways of encryption also. In this protocol, Alice wants to send some text to Bob so she decides to shift every letter in this text by three, so that: \( a \) becomes \( d \), \( b \) becomes \( e \), and so on.

For example: suppose Alice wants to send Bob the word: “cryptography.” First, she will shift by every letter by 3 positions to get the word: “fubswrjudskb.” We say that the “encryption key” is ”shift forward by 3.” Second, “fubswrjudskb” is sent across the channel to Bob.

Finally, Bob will need to decrypt “fubswrjudskb” to get the original text. Knowing that Alice shifted forward by 3 positions, Bob will shift the letters back by 3 positions; so that \( d \) returns to \( a \), and \( e \) returns to \( b \), and so on. We say that the “decryption key” is ”shift back by 3.” This means that “fubswrjudskb” becomes “cryptography.”

Note that it is assumed that Oscar can see “fubswrjudskb,” but since he doesn’t know what encryption key was used, he will not be able to reverse the process and recover the word “cryptography” unless he tries to break the protocol by performing some cryptanalysis or by trying to decrypt using every possible key. These tasks can be time consuming and difficult, if not impossible. However, the difficulty or even possibility of breaking the key changes from one protocol to another. In some cases, as in the shift cipher, this task is relatively easy. However, in other protocols, this task is not trivial. In the remainder of this paper, we will describe how it is possible to break WEP.

### 2.2 Symmetric-key and Public-key Cryptography

In this section, we define the terms “symmetric-key” and “public key” cryptography and we briefly discuss some of benefits of each.

**Definition 2.3. Symmetric-key:** the same key is used to encrypt information and decrypt it.

The type of cryptography that we are describing in this thesis is what is called symmetric-key” cryptography. The other kind of cryptography that exists is called “public-key cryptography.”
Definition 2.4. Public-key: a public key is used to encrypt information, but a private key is used to decrypt it.

The algorithms and systems described in this paper belong to the first kind. In symmetric-key cryptography, Alice and Bob share the same key to encrypt and decrypt. Moreover, the encryption and decryption keys are often the same, or the decryption key is easily derived from the encryption key.

A main drawback of this type of cryptography is that both parties are required to communicate the secret key to each other by meeting, or by sending the key with a third party carrier that they both trust. This in practice represents an additional cost with respect to private key cryptography.

In public-key cryptography, each of Alice and Bob have their own public key (for encryption), and private key (for decryption). Moreover, with the current technology, knowing the public key does not reveal anything about the private key. This means that even knowing Bob’s public key, neither Alice, nor anyone else, can determine what Bob’s private key is. However, this method allows Alice and Bob to communicate securely to each other even on insecure channels.

2.3 Stream Ciphers And Block Ciphers

In this section, we give the formal definition for the term “stream cipher” and informally define “block cipher” and we discuss how RC4 which is used in WEP is a stream cipher. Informally, a stream cipher is a symmetric-key cipher where plaintext is combined bit by bit (or byte by byte) with a stream of bits.

Definition 2.5. A synchronous stream cipher is a tuple $(P, C, K, L, E, D)$ together with a function $g$ such that the following conditions are satisfied:

1. $P$ is a finite set of possible plaintexts.
2. $C$ is a finite set of possible ciphertexts.
3. $K$, the keyspace, is a finite set of possible keys.
4. $L$ is a finite set called the keystream alphabet.
5. $g$ is the keystream generator. $g$ takes a key $K$ as input and generates an infinite string $z_1z_2\cdots$ called the keystream, where $z_i \in L$ for all $i \geq 1$.
6. For each $z \in L$, there is an encryption rule $e_z \in E$ and a corresponding decryption rule $d_z \in D$. $e_z : P \to C$ and $d_z : C \to P$ are functions such that $d_z(e_z(x)) = x$ for every plaintext element $x \in P$.

In the case of RC4, the plaintext, ciphertext, keyspace, and the keystream alphabet consist of long strings made up of zeros and ones. Therefore, $P = C = L = K = \{0, 1\}$. This is because binary is the language of computers. The cardinality of the set of all possible keys
depends on which version of RC4 one is using. The early and most common RC4 version used keys which had length 64 bits. Therefore, $|K| = 2^{64}$ possible keys. However, due to how RC4 was implemented, the first three bytes were always known which means that the actual keyspace of the 64 bit RC4 is reduced to $2^{40}$ possible keys. Also, the keystream generator function in RC4 is known as the pseudorandom generation algorithm or PRGA. This function is described in details in Section . This function is used to generate the keystream bytes which are then XORed with the data. The same function is used again for decryption because the encrypted data is again XORed with the same keystream bytes to cancel the effect of the first XOR operation.

The stream of bits is usually obtained from a pseudorandom bit generator. Therefore, the plaintext can be of any length. Here, encryption happens quickly and efficiently since the bits of the data are XORed with the bits from the pseudorandom bit generator. The Vernam cipher, also known as the one time pad, is a stream cipher. The Vernam cipher is proven to have perfect secrecy; this means that the ciphertext does not provide any information about the plaintext except for possibly the length of it. This means that the one time pad can not be broken if it used correctly.

In the Vernam cipher, the ciphertext is obtained by XORing the plaintext with a key of equal length. However, the key must be totally random and should never be used more than once to have perfect secrecy. In reality, this presents a problem since creating totally random keys is costly and difficult. RC4 tries to mimic the Vernam cipher by generating pseudorandom keys, however since the keyspace is not very large, keys were often used more than once.

Block ciphers, on the other hand, use a different approach to encryption. Informally, a block cipher is a symmetric-key cipher where plaintext is encrypted on a large scale block by block.

The plaintext comes in blocks of data that have various predetermined lengths, ranging from 32, 64 to 256 bits, or even more. Ciphertext is then obtained when each block of plaintext is encrypted using the same transformation. The DES (or Data Encryption Standard) and AES (or Advanced Encryption Standard) are just two examples of well known block ciphers.

**Definition 2.6.** Initialization vector or IV: a list of numbers, or an input of fixed size, required to start a certain algorithm.

One major consideration to take into account, when deciding which of the two methods to use, is that stream ciphers usually require an initialization vector (IV) and derive the keystream of pseudorandom bits from this initialization vector. This can lead to some serious security problems if used incorrectly. For instance, if the same initial vector is used
twice, then the same pseudorandom bit keystream will be derived each time. Therefore, to have good security, the same initialization vector must never be used more than once.

RC4 which was used in WEP is stream cipher which had a 24 bit IV. This IV was too short and this meant that the same IV would be used twice after a few thousand packets were sent with the same master key. Some attacks took advantage of this weakness and reuse of IV to obtain knowledge of the plaintext.
CHAPTER 3

WEP

In this chapter we give the definition of WEP. We also describe some of the details of WEP: choosing the IV, authentication, encryption, decryption, and confidentiality in WEP.

3.1 Definition

In this section we define WEP and describe some properties of the secret keys used in WEP.

The organization best known for developing some standards in the computer and electronics industries is known as the “Institute of Electrical and Electronics Engineers” or IEEE. This organization is also responsible for setting the standards for WEP which was and is still used in some wireless networks.

Definition 3.1. Wired Equivalent Privacy (or WEP) is a security algorithm for IEEE 802.11 wireless networks.

Definition 3.2. A wireless network refers to a network that is not connected by cables.

In a wireless network, the computers, cellphones, or other electronic devices are usually connected to other parts of the network through radio communication.

Definition 3.3. WPA, WPA2: Wi-Fi protected access and Wi-Fi protected access 2. Following WEP, these are two security protocols developed by the Wi-Fi Alliance to secure wireless computer networks.

Sometimes WEP is erroneously interpreted as “Wireless Encryption Protocol.” This is because it is in fact a protocol designed to provide wireless encryption. When WEP was designed, it was meant to provide roughly the same kind of security that one might have on wired connections. In a wired network setting, protection comes in the form of physical mechanisms; for example, access to a building might be restricted to a few trusted individuals. This limitation provides a certain type of security not necessarily possible with wireless networks where radio waves can travel through walls.

Unfortunately, it was later discovered that WEP had some major problems which meant that it had failed to accomplish the job it was created to achieve. These security flaws left wireless networks vulnerable to different types of attacks.

After it was found that WEP had many flaws, it was ultimately replaced by WPA2. WPA was used to transition between WEP and WPA2. WPA was created as an interim solution when it was discovered that WEP is not strong enough, and it was created to be
backwards compatible to work with a lot of the existing infrastructure, while researchers looked for some a different way of securing information that was stronger than WEP. WPA was basically created so that production of new systems (routers and other wireless systems) doesn’t come to a halt while researchers looked for something new.

**Definition 3.4.** In WEP, the root key (or Rk) is a secret key set up by the administrator or the person in charge of the network.

In WEP, the root key is shared by all the computers or electronic devices connected to the network. Since the root key is also used by RC4 for encryption, then if the root key is leaked then anyone with access to this information will become an authorized user, and will be able to decrypt any messages that were encrypted using this key. Packets of data are encrypted using the root key which is concatenated with an initialization vector first. For each packet, a 24-bit initialization vector (IV) is chosen. The IV concatenated with the root key yields the per packet key $K = <IV||Rk>$.

We assume that the parties wishing to communicate have already agreed on a secret key and have already exchanged this secret key. This method employed for this exchange differs with the preferences of the two parties and the resources available to them. For instance, the two parties can meet and exchange keys. They can also hire a trusted third party to transport the key from one to the other. Another option is to use the Diffie-Hellman key exchange or any similar method to accomplish this task. For more on the Diffie-Hellman key exchange, we refer the reader to [4]. One benefit for using Diffie-Hellman lies in the fact that this method allows the parties to establish a secret key without meeting and over insecure channels. The secret key is valuable since without it, neither encryption nor decryption would be possible.

**Definition 3.5.** When a vector $A$ is concatenated to another one called $B$, we shall use the symbol $<A||B>$ to symbolize this.

For example: if $A = 1011$ and $B = 011$ then $<A||B> = <1011||011>$

**Definition 3.6.** The per packet key $K = <IV||Rk>$ is the root key concatenated to an IV.

In WEP, the IV is commonly concatenated at the beginning of the root key. However, there are some exceptions to this rule where the IV is concatenated at the end of the root key.

The purpose of concatenation is to add some extra security by lengthening the password and constantly changing it. Also, since the IV changes constantly, the job of an attacker trying to discover the secret key through a brute-force attack becomes more difficult.

The first version of the WEP allowed a 40 bit root key; other versions allowed 104 bits root keys too. Some vendors additionally implemented longer root key with a length up to 232 bits. In each case, the IV having 3 bytes or, equivalently 24 bits, would be concatenated to the
root keys. This means that the actual per packet key would be of length: \(40 + 3 \cdot 8 = 64\), and 128 and 256 respectively.

The time taken for a brute force attack depends on the key size; so theoretically speaking, longer keys can resist brute-force attacks for much longer because an attacker would need to have to try more keys before getting to the correct one. A rough calculation shows that it would take about a week to crack a 40 bit key by brute-force using supercomputers. This amount of time grows to over 3 million years for a key of length 104 bits. For this reason the government had rules and regulations on cryptographic exports. For example, it used to be that you could not export any security technology from the United States with a key length of more than around 40 bits.

### 3.2 Choosing the IV

In WEP, the secret key is concatenated with an initial vector to form a longer and more secure key. In this section we describe how an initial vector is chosen. We also define weak initial vectors.

The IEEE 802.11 standard did not specify how to choose the 3 bytes in the IV. Therefore, two of the popular methods are described below. Some minor adjustments were usually made to these methods before being used.

Method 1: the IV is chosen by a pseudorandom number generator independently from all other packets sent by this station.

Method 2: the station uses a counter and always remembers the last IV used. On startup the IV counter either takes a preassigned fixed value or a random number; when a new IV needs to be chosen, the station interprets the last IV used as a number and adds 1 to this number. When the highest number is reached, the station restarts again with 0.

Since the IV is made up of 24 bits, this means that a busy network will use and reuse the same IV in the course of a few hours. Also, roughly 9000 IVs out of about 16 million IVs are called weak initialization vectors.\(^1\) Using one of these weak IVs reveals valuable information about the root key. Most of the attacks on WEP take advantage of the information leaked by the weak IVs to determine the root key.

**Definition 3.7.** Weak IV: an initialization vector with a special pattern that can be exploited to attack and obtain the root key.

In fact, a weak IV has the form \((l + 3, N - 1, X)\), where \(l\) is the byte of the key to be attacked, \(N\) is usually 256 (because RC4 works in mod 256), and \(X\) can be any value.

\(^1\)More on how these weak IVs were exploited and used, will come in later sections that explain how an attack against WEP is done
3.3 AUTHENTICATION

In this section, we discuss the authentication process used in WEP. There are two types of networks defined in the IEEE 802.11 standard. The two types are called the “ad hoc network” and the “infrastructure network.”

Definition 3.8. A wireless ad hoc network is a decentralized wireless network.

Definition 3.9. An infrastructure network is a wireless network with a central base station.

The difference between the two types is the presence of a central infrastructure that handles all the communication between the devices connected to the network. For instance, in the ad hoc network two stations communicate directly together. The reader may be familiar with “Bluetooth” which is a wireless technology standard used to transfer data over short distances. For instance, Bluetooth is often used when two people want to share some data from one phone to another. In this case, the transfer of data goes directly from one device to the other without passing through a central base station.

In the second type of network, a central access point station is used as a base station. In this type of network, a device wishing to connect to the network must be first approved by this access point. Once approved by the access point, the device can communicate with the local network, but all the data must pass through the base station.

Definition 3.10. An access point (or AP) is a base station that allows devices to connect to a network.

Definition 3.11. A client is a device (or person) wishing to access a network.

Definition 3.12. The approval step, when joining a network, is commonly known as the authentication.

During the authentication, the client tries to convince the base station about the truth of its identity. The basic idea goes roughly as follows: when a new client wants to join a network through an access point, it must be able to first prove it is who it claims to be. Also, in an ideal situation, the client would like the access point to prove its own identity as well. This phase is known as authentication. WEP defines two methods by which this can be done. The two methods are called: open system authentication and shared key authentication. The two different methods are summarized in Figures 3.1 and 3.2.

Open key authentication This method is less secure than the shared key authentication. In this method, the client sends a request to the base station and the base stations responds by automatically authenticating the client. No message interchanged during this authentication process is encrypted.

Shared key authentication As the name of this method suggests, this mode makes use of the secret root key which is only known to the legitimate parties. There are fours steps to this method, and they are described as follows:
1. First, the client sends a request to the AP requesting to join the network.
2. Second, the AP responds back to the client with a random number transmitted without encryption.
3. Third, the client now has possession of the random number and is expected to encrypt the number using the secret root key. The encrypted data is sent back to the station which tries to decrypt it.
4. Fourth, the AP compares the random number from step 2 with the recently decrypted number from step 3. If the two are equal, the authentication is successful and the client is granted access to the network.

The basic idea of this method is that a client who does not have access to the root key would not be able to authenticate himself. This is because lacking the root key, a client is not able to complete step 3 successfully, and this means that no access will be granted.

Once access is granted, the two stations can exchange messages freely. These messages are encrypted and decrypted using RC4 which uses the same root key used in the authentication process. Recall that the root key is to be concatenated with the initialization.
vectors; also recall that, the same root key is used by any other device connected to the same network.

**Definition 3.13.** The payload of a WEP frame is the message to be sent along with its CRC. In WEP, the payload is the only portion of the frame that is encrypted. Figure 3.3 provides an illustration for the payload of a WEP frame.

![Figure 3.3. 802.11 frame.](image)

**The make up of a packet** When a station sends a packet, the following steps are executed:

1. The station picks a 24 bit value called initialization vector IV.
2. The IV is prepended to the root key and form the per packet key $K = IV || Rk$.
3. A CRC-32 checksum of the message is produced and appended to the payload.
4. The per packet key $K$ is fed into the RC4 stream cipher to produce a keystream $X$ of the length of the payload with checksum.
5. The plaintext with the checksum is XORed with the keystream and form the ciphertext of the packet.
6. The ciphertext, the initialization vector IV and some additional fields are used to build a packet, which is now sent to the receiver.

In a sent packet, the additional fields contain general information about the data. For instance, this information could be about the length of the data, the time of transmission, if there are further transmissions. There are also some control frames and address frames that designate where the data has to be go. The address of the AP the packet is send from/to, the address of the destination station and the address of the source station. Other WEP parameters containing the key index can be present also. The key index is used to identify which key was used when more than one root key are in use in a network.

It is important to note that some of the fields in these packets are encrypted using RC4 while some fields are left unencrypted during transmission. The attack described later in this
thesis is possible because of the header is the first value to be encrypted, and in addition to that, the header almost never changes. In fact, the first byte of an 802.11b packet is the snap header, is almost always 0xAA.

### 3.4 WEP Encryption

WEP relies on the secret root key shared between the communicating parties to protect the data being communicated. The encryption process is explained in words in this section. Figure 3.4 also summarizes this process.

![Figure 3.4. WEP encryption.](image)

**Definition 3.14.** Let $M$ be an unencrypted message, then $C(M)$ refers to the checksum of $M$. The concatenation of the plaintext $M$ and $C(M)$ is called $P = M || C(M)$. $C$ is the encryption of $P$.

Starting with the message $M$, the client begins by finding the checksum value $C(M)$. The WEP uses cyclic redundancy check CRC-32 checksum for integrity. We note that the secret key is not used in determining the value of $C(M)$. A more detailed description of how $C(M)$ is computed will be discussed later in Section 3.6. The two pieces $M$ and $C(M)$ are then concatenated to form a plaintext $P = M || C(M)$.

Simultaneously, the client starts the process for the generation of the pseudorandom keystream bytes by initializing RC4 with the chosen key per packet key $K = IV || Rk$. RC4 consisting of two parts: KSA and PRGA, will produce a pseudorandom keystream which we will call $RC4(IV, K) := X$. An example of how the keystream is generated will be described later in Section 4.4.
Finally, the ciphertext is obtained by XORing plaintext \( P = < M || C(M) > \) with the keystream \( RC4(IV, K) \). In other terms, ciphertext

\[
C = P \oplus RC4(IV, K)
\]  

(3.1)

Lastly, the client transmits \( < IV || C > \). In words, the initialization vector IV along with the ciphertext are transmitted wirelessly together by radio waves across the channel.

### 3.5 WEP Decryption

For the sake of completion and to convince the reader that the original plaintext can be recovered once it is encrypted and transmitted, we will describe how the decryption process works in this section. Figure 3.5 also summarizes the decryption process.

In order to decrypt, one must XOR the ciphertext with the keystream generated by RC4. However, before this can happen, the receiver must be able to generate the same pseudorandom keystream bytes as the sender. This is possible because the receiver knows the root key \( K \) that was previously agreed upon. Also, since the initialization vector IV was received along with \( C \), it is guaranteed that RC4 will initialize under the same conditions as before, and so it will generate the same keystream \( RC4(IV, K) \) as earlier. Therefore, to get the plaintext, decryption is straight forward and it is only a matter of an XOR operation again:

\[
C \oplus RC4(IV, K) = (P \oplus RC4(IV, K)) \oplus RC4(IV, K)
\]

\[
= P \oplus (RC4(IV, K) \oplus RC4(IV, K))
\]

\[
= P
\]

\[
=< M || C(M) >
\]

Note that, \( RC4(IV, K) \oplus RC4(IV, K) = 0 \) because \( 0 \oplus 0 = 0 \) and \( 1 \oplus 1 = 0 \) in binary. Now if the receiver has \( P = < M || C(M) > \) and wishes to verify the integrity of the message, he will start by separating \( M \) and \( C(M) \). Then starting again with the message \( M \), he can find the checksum value \( C'(M) \).

\( C'(M) \) is the compared to \( C(M) \). If the two are identical, then the data is accurate and consistent; otherwise, an error has occurred.

### 3.6 Error Detection

To insure confidentiality, WEP relies on RC4. However, to detect errors occurring during transmission, WEP relies on an error detecting code. In this section, we describe some of the properties and weaknesses of CRC-32, the error detecting code used in WEP.

**Definition 3.15.** CRC or Cyclic Redundancy Check is an error detection code. CRC-32 is a 32-bit polynomial commonly used in WEP.
As mentioned before, WEP uses RC4 to ensure the confidentiality of the data and WEP makes use of CRC-32 checksum to confirm the integrity of the data.

Originally, the purpose of CRC-32 was to detect errors that might have occurred randomly during transmission. This means that the function could determine whether an accidental error occurred, but it was not smart enough to detect when a deliberate change was executed by a determined hacker for a mischievous reason.

The basic idea behind CRC is to use special mathematics of finite fields to check for errors. To do this, one represents the long strings of zeros and ones of the data as a polynomial: $M(x)$.

This polynomial will be then multiplied by $x^n$ where $n$ is the degree of what is called the generator polynomial.
**Definition 3.16.** The generator polynomial $G(x)$ is a polynomial used by a CRC to ensure confidentiality of data.

The polynomial used by CRC-32 is the following polynomial:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$  \hspace{1cm} (3.3)

This generator polynomial $G(x)$ is chosen carefully beforehand because it is believed to have distinct properties that make it ideal for discovering special types of errors. The math behind all this is beyond the scope of this paper. The interested reader is referred to [26].

**Definition 3.17.** The CRC value $C(M(x))$ of a message $M(x)$ is the remainder when $M(x) \cdot x^n$ is divided by $G(x)$, where $n$ is the degree of $G(x)$. In symbols,

$$C(M(x)) = M(x) \cdot x^n \mod G(x).$$

We will write $C(x)$ instead of $C(M(x))$ when there is no confusion. Also, note that

$$\frac{M(x) \cdot x^n}{G(x)} = Q(x) + \frac{C(M(x))}{G(x)}$$  \hspace{1cm} (3.4)

where $Q(x)$ is the quotient and $C(M(x))$ is the remainder when $M(x) \cdot x^n$ is divided by $G(x)$. Thus either $C(M(x)) = 0$ or $\deg C(M(x)) < \deg G(x)$.

Multiplying Equation (3.4) by $G(x)$ on both sides, we get:

$$G(x) \cdot \frac{M(x) \cdot x^n}{G(x)} = G(x) \cdot \left( Q(x) + \frac{C(M(x))}{G(x)} \right)$$

$$M(x) \cdot x^n = G(x) \cdot Q(x) + C(M(x))$$

$$M(x) \cdot x^n - C(M(x)) = G(x) \cdot Q(x)$$

$$M(x) \cdot x^n + C(M(x)) = G(x) \cdot Q(x)$$  \hspace{1cm} (3.5)

since all the calculations are done in $\mathbb{Z}_2$.

Finally, once the remainder is found, it is concatenated with the message $M(x)$ to form $< M(x)||C(M(x)) >$ which is sent across the channel. Keep in mind that

$$< M(x)||C(M(x)) > \equiv M(x) \cdot x^n + C(M(x))$$  \hspace{1cm} (3.6)

On the other end, upon receiving $< M(x)||C(M(x))>$, the receiver will separate the two parts. This is possible because the $C(M(x))$ is of fixed length. To check whether errors occurred or not, the receiver can repeat the same steps to find the CRC of the message received, then the two CRC values are compared.

Alternatively, another method for checking whether errors occurred is to consider the received $< M(x)||C(M(x))>$ as the polynomial $M(x) \cdot x^n + C(M(x))$ which is supposed
to be equal to $G(x) \cdot Q(x)$ by Equation (3.6). At this point, it suffices to check whether the received polynomial is in fact divisible by $G(x)$. If it is in fact divisible by $G(x)$, then no errors have occurred during transmission.

To summarize, the steps when checking the CRC are:

Step 1: the message polynomial is multiplied by $x^n$ where $n$ is the highest power of the generator polynomial.
Step 2: the answer obtained is then divided by the predetermined generator polynomial.
Step 3: the checksum $C(x)$ is the remainder that results from the division operation.

To illustrate the above, let us consider an example. To make it easier to understand, we will assume that the generator polynomial for this CRC is the polynomial:

$$G(x) = x^3 + x^2 + 1 \quad (3.7)$$

Since the degree of this polynomial is 3, the remainder will always a degree less than or equal to 2. This means that the polynomials obtained can be of the form

$$a \cdot x^2 + c \cdot x^1 + c \cdot x^0, \text{ where } a, b, c \in [0, 1] \quad (3.8)$$

In other terms, given a message $M(x)$, this polynomial is used to produce a 3-bit $C(M(x))$ to be appended to the message $M(x)$.

Now suppose the binary message $M = 111010$ is to be sent across a channel. $M$ can represented by the polynomial $M(x) = 1x^5 + 1x^4 + 1x^3 + 0x^2 + 1x^1 + 0x^0$.

Keeping in mind that all the calculations are done mod 2, to calculate the CRC value, we shall: first, multiply $M(x)$ by $x^3$. Second, long divide the result of $M(x) \cdot x^n$ by the generator polynomial: $G(x) = x^3 + x^2 + 1$.

Indeed, first:

$$x^3 \cdot M(x) = x^3 \cdot (1x^5 + 1x^4 + 1x^3 + 0x^2 + 1x^1 + 0x^0) \quad (3.9)$$
$$= x^3 \cdot (x^5 + x^4 + x^3 + x^1)$$
$$= x^8 + x^7 + x^6 + x^4$$

Second, long divide: $x^8 + x^7 + x^6 + x^4$ by $x^3 + x^2 + 1$ as shown in Table 3.1.

Finally, we notice that the remainder is the polynomial: $C(x) = x^1 = 0x^2 + 1x^1 + 0x^0$. This equivalent to 010 in binary representation, which is the checksum value to be appended to the original message: 111010. When this is done, this will look as follows: $< 111010 | 010 >$.

Upon receiving this message, the receiver will break the string into two pieces, where the first one will be 111010 and the second is 010. This is possible because the receiver knows the generator polynomial $G(x)$ used, so the receiver knows that the CRC function will always
yield polynomials of the form described in Equation (3.9). Therefore, he knows that the last three numbers, in the received vector, always refer to the CRC value of the message. Then, the receiver will repeat the calculation done above to compare the answer he finds with 010. If there is a match, he will know that the data was not accidentally corrupted. If there is no match, then the receiver knows that the data is corrupted so he will either try to fix the error, or if that is not possible, he will ask for a retransmission.

Alternatively, another way of checking whether an error occurred or not is to use the other method described earlier. In this method, we start with the string $<111010||010>$ and write the corresponding polynomial. The receiver can then verify that the remainder when this polynomial is divided by the generator polynomial is in fact zero. The calculation will is next. Indeed, first: $<111010||010> \Rightarrow x^8 + x^7 + x^6 + x^4 + x$.

Second: long divide: $x^8 + x^7 + x^6 + x^4 + x$ by $x^3 + x^2 + 1$ and then verify that the remainder is 0 as shown in Table 3.2.

**Theorem 3.1.** It is well known that the CRC function is linear. This means that WEP checksum is a linear function of the message. In other terms,

$$C(M(x) \oplus M'(x)) = C(M(x)) \oplus C(M'(x))$$

(3.10)

**Proof.** To prove Theorem 3.1, suppose we have two messages $M(x)$ and $M'(x)$ for which we want to calculate the checksum. Now, according to the method described above, we have to first find $M(x) \cdot x^n$ and $M'(x) \cdot x^n$, once these two quantities are determined, we have to find the remainder of each when divided by the generator polynomial $G(x)$.

Then,

$$C(M(x)) = (M(x) \cdot x^n) \mod G(x)$$

(3.11)

$$C(M'(x)) = (M'(x) \cdot x^n) \mod G(x)$$
Now,

\[ C(M(x)) + C(M'(x)) = (M(x) \cdot x^n) \mod G(x) + (M'(x) \cdot x^n) \mod G(x) \]  
\[ = (M(x) \cdot x^n + M'(x) \cdot x^n) \mod G(x) \]  
\[ = ((M(x) + M'(x)) \cdot x^n) \mod G(x) \]  
\[ = C(M(x) \oplus M'(x)) \]

which means that the function is linear.

\[ C(M(x)) + C(M'(x)) = (M(x) \cdot x^n) \mod G(x) + (M'(x) \cdot x^n) \mod G(x) \]  
\[ = (M(x) \cdot x^n + M'(x) \cdot x^n) \mod G(x) \]  
\[ = ((M(x) + M'(x)) \cdot x^n) \mod G(x) \]  
\[ = C(M(x) \oplus M'(x)) \]

\[ \text{Table 3.2. CRC of } x^5 + x^4 + x^3 + x \text{ By Method 2} \]

<table>
<thead>
<tr>
<th>( x^3 + x^2 + 1 )</th>
<th>( x^5 + x^3 + x^1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( +x^8 )</td>
<td>( +x^5 )</td>
</tr>
<tr>
<td>( +x^7 )</td>
<td>( +x^3 )</td>
</tr>
<tr>
<td>( +x^6 )</td>
<td>( +x^2 )</td>
</tr>
<tr>
<td>( x^6 )</td>
<td>( x^4 )</td>
</tr>
<tr>
<td>( x^5 )</td>
<td>( x^3 )</td>
</tr>
<tr>
<td>( x^4 )</td>
<td>( x^3 )</td>
</tr>
</tbody>
</table>

\[ \text{Definition 3.18. Spoofing occurs when controlled changes are made to a message without detection.} \]

Spoofing is one possible attack that can be employed by a mischievous person in order to obtain information that otherwise should be private. Spoofing is possible because CRC is linear. Spoofing is dangerous since an attacker can make changes to messages without being detected. With those changes, Oscar can be authenticated as a legitimate user on a network even without having access to the secret key. This is one of many benefits that Oscar can obtain by employing this attack.

One consequence of the linearity property is that it is possible to make controlled and careful changes to a ciphertext without knowing the original plaintext, and the changes can be made without raising any red flags and without being discovered. While messages are in transit, they look like long strings of zeros and ones. For instance, one might see the string \( <111010||010> \) as in the example mentioned before.

Now suppose that a mischievous Oscar wants to change the message 111010 to 101010 instead. If that change is made without changing the value 010 = \( C(111010) \), then the change will be detected as a random error. Therefore, to change the string 111010 without detection, Oscar has to change the CRC value also. Now we know from Theorem 3.1 that

\[ C(M(x) \oplus M'(x)) = C(M(x)) \oplus C(M'(x)) \]
So in this case, \( M = 111010 \) and \( \Delta = 010000 \) since \( M \oplus \Delta = 101010 \) which the value that Oscar wants. Now \( C(M) = 010 \) and this is known, also, Oscar knows the generator polynomial used, so he can calculate \( C(\Delta) \) as described earlier.

First, write \( \Delta = 010000 \) as the polynomial \( \Delta(x) = x^4 \). Second, multiply \( \Delta(x) \) by \( x^3 \) since degree of \( G(x) = x^3 + x^2 + 1 \) is 3. This gives the result \( x^3 \cdot x^4 = x^7 \). Finally, divide \( x^7 \) by \( x^3 + x^2 + 1 \) as shown in the Table 3.3.

### Table 3.3. CRC of \( x^7 \)

<table>
<thead>
<tr>
<th>( x^3 + x^2 + 1 )</th>
<th>( x^4 + x^3 + x^2 + x^1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( +x^7 )</td>
<td>( +x^6 )</td>
</tr>
<tr>
<td>( +x^6 )</td>
<td>( +x^5 )</td>
</tr>
<tr>
<td>( +x^5 )</td>
<td>( +x^4 )</td>
</tr>
<tr>
<td>( +x^4 )</td>
<td>( +x^3 )</td>
</tr>
<tr>
<td>( +x^3 )</td>
<td>( +x^2 )</td>
</tr>
<tr>
<td>( +x^2 )</td>
<td>( +1 )</td>
</tr>
</tbody>
</table>

Hence, according to the division shown in Table 3.3, \( C(\Delta) = C(x^7) = 1 \). So \( C(\Delta) = 001 \), and finally,

\[
< \Delta||C(\Delta) > = < 010000||001 >
\]  

(3.13)  

Now Oscar knows all the changes he needs to make to the original string \( < 111010||010 > \) in order to avoid detection. Indeed,

\[
(M + C(M)) \oplus (\Delta, C(\Delta)) = < 111010||010 > \oplus < 010000||001 >
\]  

(3.14)  

\[
= < 101010||011 >
\]

Receiving \( < 101010||011 > \), WEP will not detect an error. Indeed, the receiver will think that the original message was in fact \( M(x) = x^5 + x^3 + x \) which if checked, yields the same CRC value transmitted. This value is 011 which corresponds to \( x + 1 \).

Verification, step 1:

\[
x^3 \cdot M(x) = x^3 \cdot (x^5 + x^3 + x^1)
\]

\[
= x^8 + x^6 + x^4
\]

(3.15)

Step 2: Long divide: \( x^8 + x^6 + x^4 \) by \( x^3 + x^2 + 1 \) shown in Table 3.4.

From the Table 3.4, we notice that the remainder is the polynomial: \( C(x) = x + 1 \). In binary representation, this equivalent to 011 which is the checksum value that was also
Table 3.4. CRC of $x^5 + x^3 + x$

<table>
<thead>
<tr>
<th>$x^3 + x^2 + 1$</th>
<th>$x^5 + x^4 + x^2 + x^1 + 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+x^8$</td>
<td>$+x^6$</td>
</tr>
<tr>
<td>$+x^7$</td>
<td>$+x^5$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^7$</td>
<td>$+x^6$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^4$</td>
<td>$+x^3$</td>
</tr>
<tr>
<td>$+x^3$</td>
<td>$+x^2$</td>
</tr>
<tr>
<td>$+x^3$</td>
<td>$+x^2$</td>
</tr>
<tr>
<td>$+x^4$</td>
<td>$+x^3$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
<tr>
<td>$+x^5$</td>
<td>$+x^4$</td>
</tr>
</tbody>
</table>

received. This means that the receiver will be tricked into thinking that the data was not corrupted, which is of course not the case.

This shows how a careful attacker can change the data without being detected. This is of course very bad, and so the WEP checksum fails to protect data integrity. Keep in mind that data integrity is one of the three main goals of the WEP protocol.
CHAPTER 4

RC4

As mentioned in Section 3.6, to insure confidentiality, WEP relies on RC4. This chapter will introduce some of the history of RC4. We also present some of the notation that we will use when describing the algorithm, then we will provide the description of the algorithm along with an example in which we illustrate how RC4 works.

4.1 SOME HISTORY

In this section we provide some background history of RC4. We also mention another use for RC4 and finally, we give a description of the secret internal state of RC4.

The stream cipher RC4 is the encryption algorithm used in WEP. RC4 is a stream cipher designed in the RSA laboratories by Ron Rivest in 1987. This cipher is widely used in commercial applications including Oracle SQL, Microsoft Windows and the SSL. As mentioned before in Section 2.3, RC4 uses symmetric-key encryption, meaning the key is the same for encryption and decryption.

An alleged implementation of RC4 (Ron’s Code #4) was posted on September 13, 1994 anonymously on the Internet newsgroup sci.crypt cite without permission or verification from Ron Rivest.

RC4 is one of the most popular and efficient stream ciphers. It is a stream cipher with variable key length. The basic idea of RC4 is to start with the identity permutation and then use the secret key, along with an initialization vector, to produce a random looking permutation. This happens in the Key Scheduling Algorithm (KSA) phase. Based on this pseudorandom permutation which depends on the secret key, the next stage of Pseudorandom Generation Algorithm (PRGA) generates keystream bytes which get XOR-ed with the plaintext bytes to generate ciphertexts.

We now know that the RC4 stream cipher is not secure especially when one performs multiple encryptions using the same keystream. As mentioned above the stream cipher works by generating a keystream, based on the secret key, together with an initialization vector (IV). Therefore, if both the plaintext and the ciphertext are known, the keystream generated by RC4 can be recovered by simply XORing the plaintext and the ciphertext together. This is in fact
true because

\[
ciphertext = plaintext + keystream
\]

\[
ciphertext + plaintext = plaintext + keystream + plaintext = keystream
\]

since in binary: \(0 \oplus 0 = 0\) and \(1 \oplus 1 = 0\).

Interestingly, if an attacker has recovered a part of or the whole keystream, then this information can be used by that person to decrypt any subsequent ciphertext generated by the same access point even without possession of the secret root key. This is possible because many of the initialization vectors were used repeatedly, thus the same keystreams were used multiple times to encrypt. This creates another vulnerability.

Historically, due to restrictions by the US government on the exports of cryptography, a 64-bit WEP was implemented. WEP-40, as it was also known, had a key length of 40 bit. This key was to be concatenated with the 24 bit initialization vector IV to form the 64-bit secret key used in RC4. Later versions of RC4 had some larger secret keys reaching 256 bits.

The earlier versions had a short key length on purpose, since the short keys meant that WEP was deprived of some security and thus, it could be broken by brute force attacks. It was thought that if the key is long enough, typically, 128 bit (16 byte) then the encryption is strong. However, this was later shown to be false. In any case, shorter insecure key sizes have been widely used due to export restrictions and some implementation problems. Under all of these unfortunate conditions, RC4 became even less secure.

Not only was WEP blamed for having a small key size, but WEP was also criticized for lacking a specific “key management” standard. This meant that keys were short and initialization vectors were overused, which caused even further weaknesses in the system. It was later found that even if keys were to have been replaced frequently then the overall security provided by WEP would not have improved by very much.

**RC4 as a random number generator.** Another use for the RC4 algorithm is to generate random numbers. Indeed, Mark Stamp claims in his book “Information security” that this possible as long as the initial key is chosen appropriately and randomly.

How can we generate cryptographic random numbers? Since a secure stream cipher keystream is not predictable, the keystreams generated by say, the RC4 cipher, must be a good source of cryptographic random numbers. Of course, the selection of the key, which is like the initial seed value for RC4 remains a critical issue.

However, it is likely that RC4 is not a true random number generator, in the sense that the numbers in the output given are not given in a truly uniform fashion. Indeed, in [2],
Andreas Klein investigates work done by J. Dj. Golic and others. This investigation leads us to believe that RC4 is not a true random number generator. In section 2.2 of the paper [2], we read that J. Dj. Golic has shown that the first $2^{40}$ bytes of RC4 are distinguishable from a true random sequence of numbers because of a certain correlation. Work done by others in the field has decreased this bound even lower to $\approx 2^{26}$.

Finally, let us turn our attention to the secret internal state of RC4. RC4 has a secret internal state of about 1700 bits. To understand how this number is obtained, we recall that the data structure consists of an array that we will call $S$, because $S$ holds the “state” of the algorithm. This $S$ will be transformed and it will hold the final permutation of length $N$ that feeds into the PRGA at the end. $N$ is typically 256.

The final permutation is kept private. Also, the two indices $i$ and $j$ associated with the RC4 algorithm are kept private. This is all kept a secret in addition to the secret key.

Therefore, when $N = 256 = 2^8$, the array $S$ has length 256, where each position in the array can be filled with any one of 0, 1, 2, 3, ..., 255 without repetitions. This can be done in $256!$ different ways. Now, the two indices $i$ and $j$ are also kept private, and each one of them can be any one number of 0, 1, 2, 3, ..., 255. Therefore, this means that there are $256 \cdot 256 = 2^8 \cdot 2^8$ different choices for this.

Putting the two observations together, we get that the secret internal state is $\log_2(2^8! \cdot (2^8)^2) \approx 1700$ bits.

Therefore, due to the big size of the secret state (1700 bits for $N = 256$), some brute force attacks aimed at recovering the secret state become completely impractical as they require more than $2^{700}$ steps.

Next we present the RC4 algorithm. All additions in both the KSA and the PRGA are additions mod $N$ with $N = 256$ typically.

### 4.2 Notation

In this section we introduce some of the math notation that we need to describe RC4 in more details. For the remainder of this chapter, numbers are written in decimal notation. Also, the signs $+, -, \cdot$ are the addition, subtraction and multiplication mod $N$.

**Definition 4.1.** A string $S$ of numbers is called a permutation of $N$ numbers if each of the numbers from 0 to $N-1$ are all listed only once in this array.

For example: if $N = 5$, then $S = [0, 1, 2, 3, 4]$ will be called a permutation. Note that this is permutation is not the same as the permutation $S' = [1, 0, 2, 3, 4]$.

Note: the use of the [ and ] symbols might be confusing, since it might remind the reader of sets where order does not matter. However, the order matters for us here.

**Definition 4.2.** A transposition is an operation that switches two entries in a string.
For example, one transposition on \( S = [0, 1, 2, 3, 4] \) could possibly exchange the numbers 0 and 2 leaving every other bit unchanged; this would result in the transposition \( S' = [2, 1, 0, 3, 4] \).

The first element in the permutation \( S \) is always \( S[0] \). The \( i \)th element of the permutation will be called \( S[i] \) where \( i = 0, ..., N - 1 \). Also, given a permutation \( S \) then \( S^{-1} \) is considered to be the inverse permutation. That is, if \( S[i] = j \) then \( S^{-1}[j] = i \).

For example: using \( S = [0, 1, 2, 3, 4] \) and \( S' = [2, 1, 0, 3, 4] \), we can say that \( S[0] = 2 \) or that \( S^{-1}[2] = 0 \).

WEP is mostly based on the RC4 stream cipher and RC4 is based on the two algorithms: KSA and PRGA. To analyze these two algorithms, we will write the following: \( S_k \) and \( j_k \) for the state of \( S \) and \( j \) after exactly \( k \) rounds of the loop in KSA. This means that \( S_0 = [0, 1, 2, ..., 255] \) always; thus writing \( S_0[0] \) will indicate that we are referring to the number in the zeroth position of the array \( S_0 \) which is 0. Writing \( S_0[1] \), will indicate that we are referring to the number in the first position of the array \( S_0 \) which is 1, and so on.

Note that KSA will loop for exactly \( N \) times, and so \( S_{k+N} \) and \( j_{k+N} \) will be the states of \( S \) and \( j \) after KSA has ended and the PRGA has looped \( k \) times producing \( k \) bytes. In general, \( S_l \) will be understood as the permutation after round \( l \) is completed. Similarly, \( j_l \) will be understood as the \( j \) in round \( l \).

Given a specific key \( K \), the keystream produced by PRGA will be called \( X = RC4(K) \). Just as above, writing \( K[0] \), for example, will indicate that we are referring to the number in the zeroth position of the key \( K \), and so on.

Note that an attacker who knows the first \( k \) bytes of an RC4 key \( K \) also knows \( S_k \) and \( j_k \).

### 4.3 Description of the RC4 Algorithm

In this section we present the RC4 algorithm and describe the steps of the algorithm.

In \( RC4 \), the user will enter his/her preferred key which can have any length, the bytes of the secret key are denoted by \( K[0], ..., K[l-1] \). This key is then transformed into a key of length \( N \) using the formula \( K[i] = K[i \mod l] \) for any \( i \), such that \( 0 \leq i \leq N - 1 \). Basically, this formula repeats some earlier key bits to form an array \( K \) of the same size as \( N \). Typically \( N = 256 \), but there are other instances where \( N \) is chosen to be more or less than 256.

Next we present the RC4 algorithm. Note that the keystream generation process starts at the end of the PRGA. This is the keystream which will be added to the plaintext in order to get the ciphertext later on. Here we assume that all additions in both the KSA and the PRGA are additions \( \mod N \) with \( N = 256 \).

As the reader may notice, the code for RC4 as shown in Table 4.1 is very brief and doesn’t involve any complicated operations. This makes its implementation easy which is also
Table 4.1. RC4

<table>
<thead>
<tr>
<th>KSA</th>
<th>PRGA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization:</strong></td>
<td><strong>Initialization:</strong></td>
</tr>
<tr>
<td>For (i = 0, \ldots, N - 1)</td>
<td>(i = j = 0;)</td>
</tr>
<tr>
<td>(S[i] = i;)</td>
<td><strong>Keystream Generation Loop:</strong></td>
</tr>
<tr>
<td>(j = 0;)</td>
<td>(i = i + 1;)</td>
</tr>
<tr>
<td><strong>Scrambling:</strong></td>
<td>(j = j + S[i];)</td>
</tr>
<tr>
<td>For (i = 0, \ldots, N - 1)</td>
<td><strong>Swap</strong> ((S[i], S[j]);)</td>
</tr>
<tr>
<td>(j = (j + S[i] + K[i]));</td>
<td>(t = S[i] + S[j];)</td>
</tr>
<tr>
<td><strong>Swap</strong> ((S[i], S[j]);)</td>
<td><strong>Output</strong> (z = S[t];)</td>
</tr>
</tbody>
</table>

one of the beauties of this stream ciphers. However, without some knowledge in the language of computers, one may not be able to fully understand what these simple few lines mean.

Next, we will describe each of these steps in simple terms. Later, we will present a toy example with numbers to show how this algorithm would work in real life.

Before we start by the describing RC4 and its two phases, we assume that the parties communicating have already agreed on a private key and have already exchanged this secret key. Also, that the secret key has the same length as the string size in KSA.

Therefore, we start by running the KSA. The purpose of the KSA is to obtain a random permutation starting with the initial simple string: \(S_0 = [0, 1, 2, 3, \ldots, 255]\). In order to do this, we will need to perform a number of transpositions on \(S_0\). We can think of this \(S_0\) as a chain of 256 numbered boxes setup in a straight line one next to another.

Now the positions affected by the transposition, i.e.: the two bits to be exchanged are determined by the following rules: initially we start by swapping the number 0 box with the number \(x\) box. Then, we move to the first box and swap that box with number \(y\) box. This will be repeated enough times and will finally stop once 255th box is swapped. The counter \(i\) is allowed to go from 0 to 255 to keep track of which box we are swapping. In addition to that, \(i\) is used in the formula that calculates the place of the second box that is to be swapped with. This second box is called \(j\) in the algorithm above is determined according to the formula \(j \equiv (j + S[i] + K[i]) \pmod{256}\). Finally, \(i\) is increased by 1 once a swap has been completed; this way it is used to keep track of which step the KSA is on. This pattern of steps repeats until \(i\) finally reaches 256.

Let us turn our attention to dissect the formula \(j \equiv (j + S[i] + K[i]) \pmod{256}\) in order to understand how \(j\) is truly chosen. Recall the following two code lines from KSA: “\(j = 0\)” and “For \(i = 0, \ldots, N - 1.\)”

The purpose of the “\(j = 0\)” is to reset the counter \(j\) in case it was being used someplace else before KSA. The “for” loop guarantees that KSA will run the required \(N\) steps; \(N\) being the
size of the permutation $S_0$. Therefore, at every step, $j$ will be added to the number in the $ith$ position of $S$ and to the number in the $ith$ of the secret key $K$. This is addition is done modulo $N$ in order to stay within the bounds 0 and $N-1$.

In the following step, $i$ would have increased to become $i + 1$, and a new $j$ is to be calculated. To obtain this new value for $j$, we add the old $j$ from the last step with the number in the $(i + 1)th$ position of $S$ and to the number in the $(i + 1)th$ of the secret key $K$. We note that every entry of the original $S_0$ is swapped at least once during the $N$ steps above. However, it is possible that $i = j$, in which case the entry in $S[i]$ is swapped with itself.

After running the required number of iterations, the KSA yields a new permutation which is in turn fed into the PRGA to determine the keystream bits. These bits produced by PRGA will be eventually XORed with the data that is to be encrypted. On the receiving end, these same bits will be used to decrypt the messages sent.

Let us turn our attention to the PRGA part of RC4 and describe how it works. This first step is described in Figure 4.1. At the initial stage of PRGA, we start by resetting the two indices $i$ and $j$. Once this is completed, we start a new loop that will produce the bits $z_0, z_1, z_2, ...$ as needed until we have enough bits to encrypt or decrypt the message at hand. The string $z_0, z_1, z_2, ...$ will be called $X = RC4(K)$. At times, we will write $z_0 = X[0]$ to refer to the 0th keystream bit, and so on, just as we did for $S$ and $K$.

The index $i$ will be used again by the algorithm to keep track of the number of iterations done, once the iteration is complete and PRGA has produced a new keystream byte, then $i$ will increase by 1; of course, it will always reset once itself automatically once it reaches 256 because all the additions are done modulo 256. $j$, the other index, also increases at the end of every iteration according to the formula $j \equiv (j + S[i]) \mod 256$ which requires $j$ to be added to the number in position $i$ of the permutation $S$. Once this is done, $S$ itself is changed again; the change happens when we swap the boxes positions $i$ and $j$ of $S$. Those same two numbers in the boxes are added together after the swap to produce some number $t$.

Once $t$ is found, the PRGA will look up the number in box number $t$ of $S$, and that will be the byte produced by PRGA. Finally, PRGA either stops or repeats the same steps depending on whether it has produced enough bits to encrypt/decrypt all the data.

### 4.4 Simple 5-byte Example

To solidify the concepts described in the previous section, we present a toy example in this section. Due to the repetitiveness of the process, we have chosen to keep the value of $N$ small with $N-1 = 4$. The key $K$ is also chosen to be of the same size.
In the following example \( N = 5 \), and so all the calculations below are done \( \text{mod} 5 \).

Written as arrays, let us suppose that:

\[
\begin{array}{c}
K \\
\begin{array}{cccc}
1 & 5 & 2 & 3 \\
\end{array} & 7 \\
K[4] \end{array}
\]

and that

\[
\begin{array}{c}
S_0 \\
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
\end{array} & 4 \\
S_0[4] \end{array}
\]

The **KSA** part of the algorithm is next. The next 5 iterations of KSA are also described in Figure 4.2.

**First iteration of KSA:**

Set \( i = 0, j_0 = 0 \). Then calculate:

\[
\begin{align*}
 j_1 &= j + S_0[i] + K[i] \\
 &= j_0 + S_0[0] + K[0] \\
 &= 0 + 0 + 1 = 1
\end{align*}
\]
So Swap $S_0[0] = 0$ and $S_0[1] = 1$ to get:

$$S_1 = \begin{array}{ccccc}
1 & 0 & 2 & 3 & 4 \\
\end{array}$$

Second iteration of KSA:
Set $i = 1$. Then calculate:

$$j_2 = j + S_1[i] + K[i]$$

$$= j_1 + S_1[1] + K[1]$$

$$= 1 + 0 + 5 = 1$$

So Swap $S_1[1] = 0$ and $S_1[1] = 0$ to get:

$$S_2 = \begin{array}{ccccc}
1 & 0 & 2 & 3 & 4 \\
\end{array}$$

Third iteration of KSA:
Set $i = 2$. Then calculate:

$$j_3 = j + S_2[i] + K[i]$$


$$= 1 + 2 + 2 = 0$$
So Swap $S_2[2] = 2$ and $S_2[0] = 1$ to get:

\[
S_3 = \begin{array}{ccccc}
2 & 0 & 1 & 3 & 4 \\
\end{array}
\]

Fourth iteration of KSA:
Set $i = 3$. Then calculate:

\[
j_4 = j + S_3[i] + K[i] \\
= 0 + 3 + 3 = 1
\]

So Swap $S_3[3] = 3$ and $S_3[1] = 0$ to get:

\[
S_4 = \begin{array}{ccccc}
2 & 3 & 1 & 0 & 4 \\
\end{array}
\]

Fifth iteration of KSA:
Set $i = 4$. Then calculate:

\[
j_5 = j + S_4[i] + K[i] \\
= 1 + 4 + 7 = 2
\]

So Swap $S_4[4] = 4$ and $S_4[2] = 1$ to get:

\[
S_5 = \begin{array}{ccccc}
2 & 3 & 4 & 0 & 1 \\
\end{array}
\]

Notice that $S_0 = S = [0, 1, 2, 3, 4]$ has been turned into the new $S_5 = [2, 3, 4, 0, 1]$ according to the steps described in the algorithm above. These steps are shown in Figure 4.2.

Once KSA is finished the **PRGA** part of the algorithm starts. This first step of the algorithm is summarized in Figure 4.3. Taking $S_5 = [2, 3, 4, 0, 1]$ from the KSA phase, we start PRGA by resetting $i = j = 0$.

On the first iteration of PRGA, we calculate:

\[
i + 1 = 0 + 1 \\
= 1
\]

Then

\[
j_{5+1} = j_0 = j + S_5[i] \\
= j_0 + S_5[1] \\
= 0 + 3 = 3
\]
So we swap $S_5[1] = 3$ and $S_5[3] = 4$ to get:

$$S_{5+1} = S_6 = \begin{array}{cccc} 2 & 0 & 4 & 3 \end{array} \begin{array}{c} \vdots \end{array} \begin{array}{c} S_6[0] \end{array} \begin{array}{c} S_6[1] \end{array} \begin{array}{c} S_6[2] \end{array} \begin{array}{c} S_6[3] \end{array} \begin{array}{c} S_6[4] \end{array}$$

\[ i=1 \quad j=3 \]

\[ \begin{array}{cccccc} 2 & 3 & 1 & 0 & 4 \end{array} \]

\[ \begin{array}{cccc} 2 & 0 & 4 & 3 & 1 \end{array} \]

\[ 0+3=3 \]

Figure 4.3. RC4 PRGA.

Finally, the output is:

\[ z = S_6[S_6[i] + S_6[j]] \]
\[ = S_6[S_6[1] + S_6[3]] \]
\[ = S_6[3] \]
\[ = 3 \]
\[ = X(0) \]

This first iteration of PRGA is shown in Figure 4.3.
CHAPTER 5
ATTACKS ON WEP

Over the years, hackers have exploited some of weaknesses found in WEP to conduct attacks on it in order to obtain private information. This chapter investigates some attacks which are theoretical and some of which are indeed tangible.

5.1 FEASIBILITY OF ATTACKS ON WEP

Given that we know the inner workings of every stage of WEP, and given that the processing powers of computers has increased over the years, while their cost to the public has also decreased, attacking WEP has become very practicable and real. In this section we discuss the feasibility of some attacks.

One of the purposes of RC4 was insuring the privacy of information transmitted between say Alice and Bob. Therefore, information sent from Bob to Alice would be encrypted using the keystream generated by RC4.

However, due to some poor design choices and implementations, WEP was not as strong and secure as desired; this allowed Oscar, the attacker, to learn parts of the private communication between Alice and Bob. In some cases, Oscar was able to do this with minor work.

As mentioned earlier, the WEP protocol consisted of multiple parts, checksuming and encryption of the data, transmission, and decryption of data. Given this predictable protocol which used algorithms well known and studied in literature, Oscar had a assortment of choices for attacking WEP and leaking information that would have been considered private otherwise.

Since the WEP protocol was designed to safeguard data transmitted wirelessly and over radio waves, Oscar would need some instruments to capture and transmit the data over the airwaves. This was and is still not a difficult task since many over the shelve tools already exist, or can be easily modified, for just this purpose. Oscar would also need a personal computer for storage, manipulation, encryption, and decryption of data. The costs of these tools might be high, however, since routers, wireless transmitters, and laptop computers are readily available for purchase in the market, we can assume that attacks are practical and can be done with minor preparations by a malicious party.

Indeed, several computer programs that implement one attack described later (the FMS attack) exist. Some of these computer programs are: WEPCrack, AirSnort, and bsd-airtools.
WEPCrack from http://wepcrack.sourceforge.net was the first publicly released tool. AirSnort from http://airsnort.shmoo.com is much better known, mainly because it is easier to use.

Given that attacks are possible and plausible, one might wonder, what advantages one might benefit from attacking WEP in the first place? In other terms what benefits does Oscar obtain by investing his resources in this problem? The answer to this question varies with the level of damage that Oscar plans on inflicting. Oscar’s goal could be to simply eavesdrop and listen to the conversation between Alice and Bob. Oscar might want to convince Alice that he is a third party, for instance Bob or Eve. Another possibility is that Oscar might want to tweak the message that Bob is sending to Alice. With this in mind, Oscar must have a plan of attack to accomplish each goal. Some of these goals can be accomplished by attacking the CRC-32 checksum function, or the RC4 and specifically the KSA portion of the protocol.

For instance, Oscar can attack the KSA in order to determine the secret key shared by Alice and Bob. Obtaining this key allows Oscar to encrypt and decrypt information sent anytime. Sometimes, it is hard to guess the secret key, so Oscar might be content learning a portion of the keystream and decrypting only part of the data. Attacks directed at CRC-32 can give Oscar the ability to make modifications to the encrypted data without ever being detected. This message modification ability leads to very surprising results, for instance, it can “be leveraged to decrypt messages sent over the air.”

In our reading on RC4 and WEP, we have come across a few different types of security issues and flaws that have been well researched. Also, different types of attacks that take advantage of these flaws were mentioned in different papers, books, and websites. Scott Fluhrer, Itsik Mantin, and Adi Shamir are three of the most famous people attributed to have worked on some of the earliest attacks on WEP. Their work was studied by others. The attacks described by them were improved and implemented in some of the computer programs mentioned in Section 5.1. We will make an attempt to explain the attack described by them later.

5.2 List Of Attacks

In this section, we mention the names of a few attacks along with a short description of each. Some of these attacks are sensitive and can be rendered irrelevant if some changes are made to RC4 or WEP. In fact, according Mark Stamp, the author of [19], it is possible to avoid certain types of attacks on RC4 if the first 256 bits are discarded. Of course this is not a problem as long as both, Bob and Alice agree beforehand that they will do this. Otherwise, if one of them discards the information but the other one doesn’t then they will run into problems deciphering the data.
**Distinguishing Attacks.** These types of attacks are used to distinguish the RC4 pseudorandom sequence from a true random sequence.

In 1997, Golic showed that the sum of the last bits at time step $t$ and $t + 2$ is correlated to 1. Golic showed that $\approx 2^{40}$ bytes are needed before one can tell whether a sequence of numbers is a true random sequence or the keystream of the RC4. In 2000, Fluhrer and McGrew found correlations between two successive output bytes. Here, approximately $2^{30}$ bytes are sufficient to distinguish RC4 from random noise. A better distinguisher which requires $2^8$ data was described by Mantin and Shamir in [11].

**Attacks On The Keystream Generation.** In this type of attack, one simulates the generation process and keeps track of all the known values in $S$ that has been verified to be true this far. Whenever an unknown entry is needed to continue the simulation, the attacker will either try all the possible values or guesses one and continues the simulation. Note that the number of trials will normally be less than $N$ since $S$ is a permutation and so the values in $S$ cannot be repeated. The time complexity of this type of attack was analyzed by Lars R. Knuden Willi Meier, Bart Preneel, Vincent Rijmen, and Sven Verdoolaege in Abstract of Analysis Methods for (Alleged) RC4 in [18]. The complexity of one of the attacks described is estimated to be less than the time of searching through the square root of all possible initial states.

This kind of search and guess attack is simple to implement, but is very inefficient in time and completely impractical for RC4 keys larger than 4 bytes. Nonetheless, ideas from this attack are sometimes useful in other types of attacks. In fact, according to the abstract of [18] the analysis methods reveal intrinsic properties of alleged RC4 which are independent of the key scheduling and the key size. However, this still poses no threat to alleged RC4 in practical applications.

**Related-Key Attacks.** In these types of attacks, it is assumed or known that the RC4 uses a long key. The attacker can then choose a long key and compare the two keystreams generated with the two keys. Comparing the outputs, the attacker may learn some information leading to the disclosure of bits and pieces of the true secret key.

**The IV weakness of the KSA.** These are attacks that exploit the weaknesses in the key scheduling algorithm of RC4. These attacks take advantage of some specific types of initial vectors that leak information about RC4.

The FMS attack is the main attack that we will focus on and describe later. This type of attack allows someone to recover the secret key used by RC4, from a large number of messages in the encrypted stream. In fact, this type of attack was used by special agent Bicker in 2005 in a demonstration to break into a Wi-Fi network that used RC4 to encrypt data. Agent Bicker, along with a couple of helpers, was able to obtain the correct and secret WEP
key in about 3 minutes and in front of an audience using off the shelf tools and computer software. The attacks takes advantage of some of the packets format used in 802.11 where the first plaintext byte of every encrypted portion of a frame is always known. In fact, it is always “AA” denoting 802.2 LLC DSAP for encapsulation. [8, 9, 3]

**Security Flaw: Second Byte Bias.** As mentioned earlier, we know in the present time that the RC4 algorithm is not secure. The research shows that RC4 has certain properties that make it less than ideal random number generator. In fact, it has been shown that one can distinguish the keystream sequence from a totally random sequence of numbers. In a random sequence of numbers, the probability of a certain number showing up is equally likely. We say that the number show up in a uniform distribution. However, research exists showing that the second byte in the output of RC4 is biased. A substantial amount of the research on this was done in [11]. The main observation is that the second output of RC4 tends to take on the value 0, with twice the expected probability of any other number. When $N = 256$, this means that the probability of the second byte taking on the 0 value is $\frac{1}{128}$ instead of the expected probability of $\frac{1}{256}$.

On a related note, in [10] Itsik Mantin writes:

> Andrew Roos noted that keys which have $K[0] + K[1] \equiv 0 \pmod{N}$, the first output is equal to $K[2] + 3$ with probability $2^{-2.85}$.

The cryptanalyst can use this fact to deduce two bytes of information about the key ($K[0] + k[1]$ and $K[2]$) with probability $2^{-10.85}$ instead of the trivial $2^{-16}$ and thus reduce the effective key length in exhaustive search by about five bits.

Grosul and Wallach showed that for large keys whose size is close to $N$ bytes, then similar keys produce similar initial states and similar streams. This means that for large Keys, the RC4 is vulnerable to related key attacks.

However, there does not seem to be any attacks on WEP that take advantage of this bias in the second byte.
CHAPTER 6
THE FLUHRER-MANTIN-SHAMIR ATTACK

One of the earliest and most damaging attacks on WEP is known as the Fluhrer - Mantin - Shamir attack. Many other attacks are based on the attack described by these three cryptographers. In this chapter, we discuss introduce the attack explaining what makes this attack so successful. Also, we demonstrate how the attack works by providing various examples. Finally, we end by briefly indicating some of the countermeasures taken to defeat this attack.

6.1 INTRODUCTION

One of the most devastating attacks on WEP was first described by the three cryptographers: Scott Fluhrer, Itsik Mantin, and Adi Shamir, and so accordingly, the attack they described is known as the FMS attack [28]. Note that the attack was described by these three cryptographers from the theoretical point of view; however, they did not test the attack in real life. The basis of the attack is a small flaw in the key scheduling algorithm (KSA) of the RC4 encryption protocol, on which WEP is based. The three cryptographers noticed a certain statistical anomaly that arose when certain keys with special forms were used. Given the structure of these special keys, it was observed that a small portion of the key ended up in the keystream more often than the remainder of all the other values. Although this flaw sounds insignificant, cryptographers are known to seize these kinds of nonrandom anomalies and turn them into opportunities and attacks to break cryptographic algorithms.

The attack described by Fluhrer, Mantin, and Shamir showed that there is a 5% probability that the values held in $S[0], ..., S[3]$ will NOT change after the first three iterations of the KSA. In other words, the values in $S[0], ..., S[3]$ will not change after the third iteration of KSA; and they will appear in the same positions on the permutation that will be used again in the first iteration of the PRGA. In other words, any hacker can guess what will happen during the KSA process with a 5% likelihood of being correct.

The attack also takes advantage of a deficiency in the design of WEP. Due to the requirements of WEP, the first value to be encrypted is always the SNAP header. This value is well known to the public and never changes. In fact, the value equals AA in hex or 170 in decimal form. This essentially means that a hacker can obtain the first byte of encrypted text and then XOR it with 170. The hacker can then deduce the first output byte of the PRGA.
In the WEP encryption process, it has been determined that the initialization vectors with the special format: \((l + 3, 255, X)\) leak information about the key. These weak IVs can be used to attack WEP and deduce the key. \(l\) is the byte of the key to be attacked. The 255 value indicates that the KSA is at a vulnerable point in the algorithm. \(X\) can be anything.

Now since IVs are joined with the secret root key to make the key, and these same IVs are sent in the clear during all the communication, then an attacker is assumed to know the first three bytes of the key, in other terms, Oscar has knowledge of \(K[0]\), \(K[1]\) and \(K[2]\). Oscar now wants to learn the 4th byte, so \(l = 0\) on the first iteration of the attack. Once Oscar learns the 4th byte, he will set \(l = 1\) and repeat the same attack to learn the 5th byte of the key. This is repeated enough times until the root key is found.

Now since the attack relies on keys with special format, Oscar must first gather large amounts of encrypted data before he can actually apply the attack. Once the gathering of data is finished, Oscar has to sort through it all, keep the “interesting” packets of data which were encrypted using the special keys that were concatenated with weak IVs, and discard the rest of the packets.

The *FMS* attacks claims that there only about 5% chance that one byte of the key will be leaked in these interesting packets. Therefore, Oscar must gather hundreds of packets and keep statistics counting the number of times each byte showed up. Over time, the leaked byte value shows up more than the other possible values. This value showing up most frequently will be Oscar’s best and most reasonable guess for the actual byte of the real key. This assurance is made higher when more packets are gathered and more statistics is done. Depending on the way the IVs were generated and the length of the key, the attack will yield successful results after about 1 million packets are gathered. In reality, 40-bit WEP can be cracked with as few as 300000 packets and 104-bit WEP can be cracked with 1000000 packets with the technology that was available in the time these attacks were first being studied.

The estimated amount of time that Oscar needs to collect packets and run the attack is about as few as 9 minutes on a saturated network. However, cracking a key is more luck than science; as mentioned earlier, Agent Bicker, with the FBI, was able to obtain the correct and secret WEP key in about 3 minutes and in front of an audience using off the shelf tools and computer software.

**Analysis of IVs generated by a little Endian Counter:** if the IVs are generated by a multibyte counter in little endian order and hence the first byte of the IV increments the fastest, then the attacker can search for IVs of the form \((l + 3, 255, X)\) for \(0 \leq l < 5\). if he can collect these for 60 different values of \(X\) then he can derive the secret key with little work. This requires approximately 4000000 packets.
Analysis of IVs generated by a big Endian Counter: if the IVs are generated by a multibyte counter in big endian order and hence the last byte of the IV increments the fastest, then the attacker can as above search for IVs of the form \((l + 3, 255, X)\). This requires approximately 1000000 packets to collect the requisite IVs.

On a slow network, capturing the required number of weak IVs can take some time. To accelerate the attack, the hacker can inject a captured WEP frame back into the network to generate more traffic. This takes advantage of the fact that WEP has no “replay protection” mechanism to prevent this. Meaning that, WEP has no problem accepting multiple packets with same IV because it was designed to do so in the first place. This means that Oscar can replay packets without raising any alarms; and WEP will never detect that it is being attacked [23]. An injection rate of 512 packets per second generally results in the required number of IVs being captured between 10 min for 40-bit and 30 min for 128-bit WEP. If no client is present on the network to generate traffic that can be captured and reinjected, in most cases the Oscar’s own computer can be made to do so.

6.2 INTO THE ATTACK

Assume that an attacker knows the first three bytes of the key \(K\) are 3, 15 and 8. Usually, these three first values are known because they are indeed the values of the initialization vector, which are sent in the clear in a packet. Knowing these values means that the attacker will be able to run the KSA for three rounds, thus learning in the process \(S_2\) and \(j_2\). We will illustrate this by using a 16-byte example with the following setup. We simulate a few iterations of the RC4 algorithm from the point of view of Oscar the attacker.

Suppose that Oscar knows the first three bytes of the key only, then \(K = [3, 15, 8, ?, ?, ?, ...]\) and \(S_0 = [0, 1, 2, ..., 15]\) are assumed true.

Written as arrays we can see that:

\[
K = \begin{array}{cccccccc}
3 & 15 & 8 & ? & ? & ... & ? \\
\end{array}
\]

and that

\[
S_0 = \begin{array}{cccccccc}
0 & 1 & 2 & 3 & ... & 15 \\
\end{array}
\]

First iteration of KSA:
Set \( i = 0, j_0 = 0 \). Then calculate:

\[
\begin{align*}
    j_1 &= j + S_0[i] + K[i] \\
        &= j_0 + S_0[0] + K[0] \\
        &= 0 + 0 + 3 \\
        &= 3
\end{align*}
\]

So Swap \( S_0[0] = 0 \) and \( S_0[3] = 3 \) to get:

\[
S_1 = \begin{array}{cccccccc}
    3 & 1 & 2 & 0 & \ldots & 15 \\
\end{array}
\]

Second iteration of KSA:
Set \( i = 1 \). Then calculate:

\[
\begin{align*}
    j_2 &= j + S_1[i] + K[i] \\
        &= j_1 + S_1[1] + K[1] \\
        &= 3 + 1 + 15 \\
        &= 3
\end{align*}
\]

So Swap \( S_1[1] = 1 \) and \( S_1[3] = 0 \) to get:

\[
S_2 = \begin{array}{cccccccc}
    3 & 0 & 2 & 1 & \ldots & 15 \\
\end{array}
\]

Third iteration of KSA:
Set \( i = 2 \)
Then calculate:

\[
\begin{align*}
    j_3 &= j + S_2[i] + K[i] \\
        &= 3 + 1 + 8 \\
        &= 13
\end{align*}
\]

So Swap \( S_2[2] = 2 \) and \( S_2[13] = 13 \) to get:

\[
S_3 = \begin{array}{cccccccc}
    3 & 0 & 13 & 1 & \ldots & 15 \\
\end{array}
\]
Note that Oscar can not run the KSA for another iteration since that would require the knowledge of:

\[ j_4 = j + S_3[i] + K[i] \]
\[ = 13 + 1 + K[3] \]
\[ = 14 + K[3] \] (6.4)

Since Oscar does not know the fourth byte \( K[3] \) of the key, he can not calculate the value of \( j \). If he knew the value of \( j \) was, then he would have to swap \( S_3[3] = 1 \) and \( S_3[j_4] = S_3[14 + K[3]] \). Currently, \( S_3[14 + K[3]] \) is unknown to Oscar.

In the attack described below, the goal of Oscar will be to determine this fourth byte of the key. Once it is obtained, he will be able to run the KSA for another iteration.

If he is successful at doing this enough times, Oscar can learn the final permutation (or state) of the KSA. Learning the final state of KSA means that the attacker will be able to produce the keystream \( X = RC4(IV, K) \) Using this keystream, the attacker is able to decrypt or encrypt data as long as the victim keeps using the same key.

\[ S_3 = \begin{array}{cccccc}
3 & 0 & 13 & 1 & \ldots & \text{"randomvalue"} \\
\end{array} \ldots 15 \]

becomes

\[ S_4 = \begin{array}{cccccc}
3 & 0 & 13 & \text{"randomvalue"} & \ldots & 15 \\
\end{array} \]

Note that in the example above, \( S_3 = [3, 0, 13, 1, 5, 6, 7, 8, 9, 10, 11, 12, 2, 14, 15] \) is the last known state of KSA from the point of view of Oscar.

**Theorem 6.1.** The FMS attack will be successful if the following two conditions hold:
First, the later iterations of KSA do not swap \( S_3[0] = 3 \) and \( S_3[1] = 0 \) and \( S_4[3] = \text{"randomvalue"} \).
Second, the attacker can learn and determine the first byte of the keystream somehow.

**Proof.** Given these two conditions why would the attack be successful? If no swaps were to happen to \( S_3[0] = 3 \) and \( S_3[1] = 0 \) and \( S_4[3] = \text{"randomvalue"} \) then the first step of PRGA would go as follows:

Recall

\[ S_{16} = \begin{array}{cccccc}
3 & 0 & ? & \text{"randomvalue"} & \ldots \\
S_{16}[0] & S_{16}[1] & S_{16}[2] & S_{16}[3] & \ldots \\
\end{array} \]
First iteration of PRGA:
Set $i_{16} = j_{16} = 0$, then set $i_{17} = i_{16} + 1 = 1$, then calculate:

$$j_{16+1} = j_{17} = j_{16} + S_{16}[i]$$

$$= j_{16} + S_{16}[1]$$

$$= 0 + 0$$

$$= 0$$

Now swap $S_{16}[i] = S_{16}[1] = 0$ and $S_{16}[j] = S_{16}[0] = 3$ to get:

$$S_{16+1} = S_{17} = \begin{bmatrix}
0 & 3 & ? & "randomvalue" & \\
S_{17}[0] & S_{17}[1] & S_{17}[2] & S_{17}[3] & 
\end{bmatrix}$$

And the first keystream output byte would be calculated as follows:

$$z = S_{17}[S_{17}[i] + S_{17}[j]]$$

$$= S_{17}[S_{16}[j] + S_{16}[i]]$$

$$= S_{17}[S_{16}[S_{16}[1]] + S_{16}[1]]$$

$$= S_{17}[S_{16}[0] + 0]$$

$$= S_{17}[3]$$

But by the assumption of the theorem, we know that no swaps affected the third component of $S$, therefore $S_{17}[3] = S_{4}[3]$. In other terms,

$$z = S_{4}[3]$$

$$= S_{3}[j_{4}]$$

$$= "randomvalue"$$

because "randomvalue" was the content of the $j_{4}th$ position of the $S_{3}$, thus

$$S_{3}[j_{3} + K[3] + S_{3}[3]] = X(0)$$

This means that on one hand, we dont know the mysterious "randomvalue" just from running the begining of KSA. But on the other hand, we can actually obtain the $X(0)$ by assumption 2 of the theorem. The value $X(0)$ is obtained from the header of the packet. In WEP, the first value of the encrypted data is always the $SNAP$ header. This header always
equals AA in hexadecimal or 170 in decimal form. This essentially means that by examining the first byte of encrypted text and XORing it with 170, Oscar can deduce $X(0)$ which the first output byte of the PRGA.

Finally, the missing key byte $K[3]$ is equal to $S_3^{-1}[X(0)] - j_3 - S_3[3]$.

$$z = X(0) = "\text{randomvalue}"$$

$$= S_{17}[3] = S_4[3]$$

$$= S_3[j_4]$$

However, $j_4 = 14 + K[3]$, so $X(0) = S_3[14 + K[3]]$ which implies that

$$S_3^{-1}[X(0)] = S_3^{-1}[S_3[14 + K[3]]]$$

$$S_3^{-1}[X(0)] = 14 + K[3]$$

$$S_3^{-1}[X(0)] = j_3 + S_3[3] + K[3]$$

In other terms, $K[3] = S_3^{-1}[X(0)] - j_3 - S_3[3]$, just as predicted. 

\[ \square \]

### 6.3 Additional Details and Efficiency

The FMS attack as published by Fluhrer, Mantin, and Shamir in 2001 was the first key recovery attack against RC4, and the attack can be generalized for WEP like operating modes.

Notice that this attack does not guarantee the value of $K[3]$ will be the correct outcome on the very first try. Depending on the given IV, the three positions mentioned in Theorem 6.1 could have been swapped in later iterations of RC4. Since Oscar can not guarantee that so swaps have occurred or not, the attack he’s employing will only give a suggestion for this value. Because the value for $K[3]$ is not guaranteed, the attack must be repeated on multiple different messages. This will yield a few different possible candidates for $K[3]$. Oscar will save the statistical data showing how many times each candidate was returned. The value for $K[3]$ that is generated significantly more often than any other is most likely the correct value. Once the value for $K[3]$ is identified, the attack can be repeated on the next sequential byte of the key.

According to Fluhrer, Mantin, and Shamir after checking about 60 weak IVs the probability of success in guessing the byte $K[3]$ should be more than 50%.

This complexity of the attack grows linearly with the as the key length increases because key bytes are recovered individually. This means that longer key lengths do not impact the delay required in this attack terribly. Also, as the reader can see from the example above, the computational effort is not significant.

There are even ways to increase the efficiency of the attack. In [25], Rafik Chaabouni describes the “Korek” attacks in detail and he also introduces a new one. KoreK is a
mathematician whose real name and identity is not known. His attacks became known after he posted them on the website netstumbler [15]. Among other ideas, KoreK suggested to check if $S_i[1] = X[0]$ or $S_i[S_i[1]] = X[0]$, which would indicate that $S_i[1]$ or $S_i[S_i[1]]$ has been modified in the remaining $RC4 - KSA$ in which case, Oscar should abandon the corresponding attack and try for another IV value. Recall that we assumed $S_3[0] = S_{16}[0] = 3$ and $S_3[1] = S_{16}[1] = 0$ in Section 6.2. This requirement then forces $X(0) = S[S[0] + S[1]] = S[3 + 0] = S[3]$.

Therefore, $X(0)$ will not be able to take on the values in the position $S_{16}[0]$ or $S_{16}[1]$ because $S_{16}[3]$ can not be either 3 or 0, since $S_{16}[0] = 3$ and $S_{16}[1] = 0$ already. Also since $S_{16}$ is a permutation, 3 can not show up in the zeroth position and then again in the third position; neither can 0.

In other terms, $X(0) \neq 3$ and $X(0) \neq 0$. This is important to note because this information can help the attacker make the attack more efficient. By checking whether $X(0) = S_3[0] = 3$ or $X(0) = S_3[1] = 0$ he can decide whether to continue with the attack or whether to ignore this specific IV and try a different one.

Why? Let us assume for a moment that that $X(0) = S_3[0] = 3$ then this would mean that $X(0)$ which is $S[S_{16}[0] + S_{16}[1]]$ is also 3. For this to happen $S_{16}[0]$ and $S_{16}[1]$ must add up to 0; because only $S_{16}[0] = 3$.

So we have a possible contradiction unless $S_{16}[0]$ was not equal to 3 in the first place. This means we get a contradiction unless $S_{16}[i] = 3$ for some $i \neq 0$. This leads us to the conclusion that $S_3[0] = 3$ was disturbed and moved around in later iterations of KSA.

Same logic applies to the case when $X(0) = S_3[1] = 0$ which will give that $X(0) = 0$ implies that $S_3[1]$ was moved in later iterations of KSA and so it is not longer equal to $S_{16}[1]$. In these two cases, Oscar will abandon the attack, and retry for another IV.

### 6.4 Formal Proof

In this section, we look at the formal proof showing that the FMS can be successful. The following information was borrowed from [6].

The following description of the attack is based on the FMS attack described in [28] however this version of the attack incorporates some of the ideas suggested by Stubblefield and KoreK. The assumptions are similar to the assumptions of Section 6.2. Therefore, we assume that an attacker knows the first keystream byte from the output of KSA. Also, the attacker knows the first $l$ words of a RC4 key and wants to attack the $l$th byte of the key for $l \geq 2$.

With this in mind, the attacker can now simulate the first $l$ steps of the PRGA and therefore we assume that Oscar is able to determine the values: $i, j_l, S_l$. 

Let’s assume that the following conditions are met:
Condition 1: \( S[l] < l \). Condition 2: \( S[l] + S[S[l]] = l \). Condition 3: \( S^{-1}[X[0]] \neq 1 \). And finally, condition 4: \( S^{-1}[X[0]] \neq S[l] \).

Now recall that the first word of the PRGA output is always

\[
X[0] = S_{N+1} \left[ S_{N+1}[1] + S_{N+1}[S_N[1]] \right]
\]

(6.10)

Therefore, if none of \( S[l], S[S[l]], \) or \( S[l+1] \) participated in any further swaps in the rest of the KSA, then the output will be

\[
X[0] = S_{N+1} \left[ S_{N+1}[1] + S_{N+1}[S_N[1]] \right] = S[l][j_{l+1}]
\]

(6.11)

However, \( S[l][j_{l+1}] = S[l][j_l + K[l] + S[l]] \). So in other words, if none of these swaps occur, then:

\[
X[0] = S[l][j_{l+1}]
\]

(6.12)

\[
= S[l][j_l + K[l] + S[l]]
\]

And solving for \( K[l] \) we get that

\[
K[l] = S^{-1}[X[0]] - j_l - S[l] \]

(6.13)

To verify this, let us assume that \( S[l], S[S[l]], \) and \( S[l+1] \) did not participate in any further swaps in the remainder of KSA. Therefore, on the first steps of the PRGA, \( i = 1 \) and \( j = S_N[1] \) but given our assumption of no swaps, then \( j = S_N[1] = S[l] \). If the first swap in the PRGA does not affect the sum \( S[1] + S[S[1]] \) nor \( S[S[1]] + S[S[1]] \), then the output will be \( S[l][j_{l+1}] \) which is equal to \( S[l][j_l + K[l] + S[l]] \). At this point, we solve this equation for \( K[l] \).

Now the first step of the PRGA will always swap \( S[1] \) and \( S[S[1]] \), but this does not affect the sum \( S[1] + S[S[1]] \). The only possibility for the first swap to affect the output of the first word would be, if \( S[S[1]] + S[S[1]] \) would be exchanged with another value, which can only happen, if \( S[1] + S[S[1]] = 1 \) or \( S[1] + S[S[1]] = S[1] \). We will check both cases separately.

Case 1: \( S[1] + S[S[1]] = 1 \). We know that this case can never happen because we assumed that all these values did not participate in any swaps in the remaining KSA. Therefore, saying \( S[1] + S[S[1]] = 1 \) is equivalent to saying that \( S[l] + S[S[l]] = 1 \). However, we already have that \( S[l] + S[S[l]] = l \) and \( l \geq 2 \).
Case 2: $S[1] + S[S[1]] = S[1]$. This also can not happen. Again, saying $S[1] + S[S[1]] = S[1]$ equivalent to saying that $S_l[1] + S_l[S_l[1]] = S_l[1]$ because we assumed no swaps occurred. By subtracting $S_l[1]$ from both sides of the equation, we get that $S_l[S_l[1]] = 0$ must hold for this. However, since $S_l[1] < l$ and $S_l[1] + S_l[S_l[1]] = l$ hold, we get that $S_l[S_l[1]] \geq 1$ and so $S_l[S_l[1]] \neq 0$.

Now if the output $X[0]$ happens to equal $S_l[1]$ or $S_l[S_l[1]]$, this would indicate that $S_{l+1}[l]$ took either the value $S_l[1]$ or the value $S_l[S_l[1]]$. This would mean that $S_l[1]$ or $S_l[S_l[1]]$ was modified after step $l$ of the KSA. But since our assumptions require that $S_l[1]$ and $S_l[S_l[1]]$ to remain unchanged after step $l$, we check for these conditions and abandon the attack if condition 3 or 4 are met.

Finally, it remains to check with which probability none of these three values is swapped in the remaining steps of the KSA. These are the steps, or the iterations, that the attacker can not see. $S[k]$ will only be swapped if either $i$ or $j$ takes the value $k$. At this point, since $i$ is increasing on every iteration, $i$ can only take values from $l + 1$ to $N - 1$. Because $l \geq 2$, $i$ will never equal 1, so $S_l[1], S_l[S_l[1]]$, and $S_{l+1}[S_l[1] + S_l[S_l[1]]]$ will not be swapped by $i$ in the remaining KSA, because $S[1] \leq l$ and $S_l[1] + S_l[S_l[1]] = l$.

The only possibility that one of these values will be swapped in the remaining $n - l$ KSA steps is, that $j$ takes the value $1, S_l[1],$, or $S_l[1] + S_l[S_l[1]]$.

In general, we can assume that $j$ takes values it values randomly from all possible values. This means that $j$ has uniform distribution.

Assuming that all three values $1, S_l[1],$, and $S_l[1] + S_l[S_l[1]]$ are different, then the probability that $j$ does not take one of these three values in one step is equal to $\frac{N - 3}{N}$.

Therefore, the probability that $j$ does not take one of these three values in all remaining steps is $\left(\frac{N - 3}{N}\right)^{N-l}$. In the special case when $N = 256$ and $l = 3$, this approximately 5.07%.

### 6.5 DEMONSTRATING THE ATTACK

In this section we show an example of how the attack would work. In this example, we assume the key and the $S$ permutation used in RC4 are both of length 256. This means that KSA will have 256 iterations. The secret root key is 52, 210, 231, 155, 224, 66, 75, 123, 99, 100, 123, 43, 213, and this will be concatenated with an IV of length 3, then the bytes will be repeated enough times so that we have a key of length 256. The results shown here can be verified using the code in Appendix (A).

Assume that RC4 with the key $K = 3, 255, 174, 52, 210, 231, 155, 224, 66, 75, 123, 99, 100, 123, 43, 213$. Also, suppose that the attacker Oscar knows the first $l = 3$ bytes of the key $K$ only. Oscar knows the values 3, 255, 174 because these are the values of the IV used. Oscar is currently interested in finding out the value $K[3]$ which we know to be 52.
Next we look at RC4 from the point of view of Oscar. Written as arrays we can see that:

\[
K = \begin{bmatrix}
3 & 255 & 174 & ? & \ldots & ? \\
\end{bmatrix}
\]

and that

\[
S_0 = \begin{bmatrix}
0 & 1 & 2 & 3 & \ldots & 255 \\
\end{bmatrix}
\]

On the first iteration of KSA, we set \( i = 0, j_0 = 0 \). Then calculate:

\[
\begin{align*}
\hat{J}_1 &= j + S_0[i] + K[i] = j_0 + S_0[0] + K[0] \\
&= 0 + 0 + 3 = 3
\end{align*}
\]

So we swap \( S_0[0] = 0 \) and \( S_0[3] = 3 \) to get:

\[
S_1 = \begin{bmatrix}
3 & 1 & 2 & 0 & \ldots & 255 \\
\end{bmatrix}
\]

On the second iteration of KSA, we set \( i = 1 \). Then calculate:

\[
\begin{align*}
&= 3 + 1 + 15 = 19
\end{align*}
\]

So we swap \( S_1[1] = 1 \) and \( S_1[3] = 0 \) to get:

\[
S_2 = \begin{bmatrix}
3 & 0 & 2 & 1 & \ldots & 255 \\
\end{bmatrix}
\]

On the third iteration of KSA, we set \( i = 2 \). Then calculate:

\[
\begin{align*}
&= 3 + 2 + 174 = 179
\end{align*}
\]

So we swap \( S_2[2] = 2 \) and \( S_2[179] = 179 \) to get:

\[
S_3 = \begin{bmatrix}
3 & 0 & 179 & 1 & \ldots & 255 \\
\end{bmatrix}
\]
Now if we are Oscar, then we can not run the KSA for another iteration since that would require the knowledge of:


(6.17)

So at this point, Oscar will assume that \( S_3[0] = 3 \) and \( S_3[1] = 0 \) will not get swapped for the remainder of the KSA, and this is in fact a true assumption.

So Oscar will start the PRGA under the assumption that

\[ S_{255} = \begin{array}{c|c|c|c}
3 & 0 & \cdots & \cdots \\
S_{255}[0] & S_{255}[1] & \cdots & S_{255}[255]
\end{array} \]

However, Oscar knows that the output of PRGA is 232 from the XOR operation with the header. Oscar now calculates

\[ K[3] = S_3^{-1}[X(0)] - j_3 - S_3[3] \mod 256 \]
\[ = 232 - 179 - 1 \mod 256 \]
\[ = 52 \]  

(6.18)

Therefore, 52 will be his guess for the key byte. Oscar will not be sure about this value being the real one, until he receives 52 after checking multiple IVs.

For instance, after checking IV : 3, 255, 158, and knowing that first keystream byte is 216. Oscar will determine that \( j_3 = 163 \) and \( S_3[3] = 1 \) so he will predict that \( K[3] = 216 - 163 - 1 \mod 256 = 52 \), again.

Doing this for all the IV of form \((3, 255, X)\) where \( 0 \leq X \leq 255 \) and keeping track of the statistics, Oscar will notice that 52 was in fact returned as a candidate 11 times, whereas every other value was voted for 3 times at most. He will then assume that \( K[3] = 52 \) and proceed to attack \( K[4] \).

In the remainder of this section, we look at a special cases where the attack fails to reveal the correct keybyte immediately. Suppose that the root key is still: 52, 210, 231, 155, 224, 66, 75, 123, 99, 100, 123, 43, 213. Also, suppose that Oscar knows that the \( IV = 3, 255, 170 \). We also suppose that Oscar can determine the first keystream byte which happens to be 99.

After doing the first 3 steps of KSA, Oscar knows that \( j_3 = 175 \) and \( S_3[3] = 1 \). At this point, he will assume that \( S_3[0] \) and \( S_3[1] \) were not swapped. We know that this assumption is bad, but Oscar does not. Finally, Oscar will calculate his prediction as

\[ K[3] = 99 - 175 - 1 \mod 256 = 179 \]
His prediction is 179, which is not the correct key byte 52 that he is after.

Using another $IV = 3, 255, 209$ and the first keystream byte of 41. Oscar knows that at $j_3 = 214$ and $S_3[3] = 1$. Therefore, his prediction will be

$$K[3] = 41 - 214 - 1 \mod 256 = 82$$

We know that this is not the correct key value. Keeping track of all the votes, Oscar will notice that 52 was selected as candidate 11 times, whereas 179 and 82 received one vote each. In fact, every other value received 0, 1 or 2 votes at most, with the exception to 180 which got 3 votes. On the other hand, 52 received 11 votes. Oscar will then assume that $K[3] = 52$ and proceed to attack $K[4]$.

### 6.6 Countermeasures

As mentioned earlier, Fluhrer, Mantin, and Shamir presented an attack against $RC4$. This attack was suspected to be applicable to WEP. This was later demonstrated true by Stubblefield. However, as described in previous sections, this attack requires the collection of the special weak $IV$s, otherwise, the attack would not yield successful results.

Faced with this attack, vendors reacted by avoiding the use of the weak IVs and filtering the weak IVs that would fulfill the requirements for the resolved condition. However, it was demonstrated that this countermeasure would not be adequate. In 2004, multiple new attacks were posted by KoreK. These new attacks were statistical attacks against WEP and did not require the weak IVs in order to be successful. Furthermore, the number of frames needed was reduced to about 500000 packets. Continued research by Klein [1] showed more improved ways of attacking $RC4$ using related keys attacks. These attacks also did not require the use of IVs and required significantly less packets.

An attempt made by Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin, demonstrated an active attack on the WEP which was much more efficient than FMS. In the abstract of [7], the authors write:

We demonstrate an active attack on the WEP protocol that is able to recover a 104-bit WEP key using less than 40,000 frames with a success probability of 50%... The IV of these packets can be randomly chosen... On a IEEE 802.11g network, the number of frames required can be obtained by re-injection in less than a minute.
CHAPTER 7
CONCLUSION

In conclusion, we echo the warnings of the experts in the fields who recommend moving away from WEP and using the currently more secure WPA2. Unfortunately however, even though it has been proven to be less than ideal, WEP still enjoys some use today. For example, the site http://wigle.net/enc-large.html has about 85356393 unique wifi networks in its database, and according to the site, about 25% of those networks still use WEP encryption. On the other hand, 40% of networks use WPA or WPA2 for encryption. WPA and WPA2 are considered to be more secure than WEP, but each one still has its own flaws.

WPA which replaced WEP improved on WEP by employing the “temporal key integrity protocol (TKIP).” The temporal key integrity protocol is a security protocol designed to replace WEP without requiring the replacement of hardware. In WEP, RC4 uses a fixed and manually entered encryption key to generate the keystream. In WPA, the TKIP is used to automatically generate a 128-bit key for RC4. In addition to that, TKIP changes the key for every packet in order to prevent attacks similar to the ones used before to break WEP. The message integrity check used by WPA was also changed and designed an attacker from capturing, altering and/or resending data packets. This new algorithm which replaced CRC32 was named Michael and was considered to be much stronger than CRC32; however, even Michael was replaced by a stronger algorithm in WPA2.

To read more on the strengths and weaknesses of Michael, the reader can refer to 802.11 Wireless Networks: The Definitive Guide by Matthew Gast [21].

WPA2 is considered to be the replacement of WPA and it was approved in 2004. WPA and WPA2 have a similar key management protocols, but one of the major differences between WPA2 and WEP or WPA is that WPA2 is a block cipher. In fact, the advanced encryption standard block cipher is used in WPA2. This encryption algorithm is considered to provide government grade security.

AES, also known as the Rijndael algorithm, is considered to be a secure and fast symmetric cipher which is easily implemented in hardware. It is worth mentioning that AES is an iterated cipher which essentially means that AES uses substitutions and permutations on every step of the algorithm and repeats this step 12 times. The key lengths allowable by AES vary between 128, 192 or 256 bits. However, these advantage means that consumers have to buy new hardware to benefit from the added security, since old hardware can not handle the
demands of AES and can not be simply upgraded with firmware updates. For a more descriptive analysis and information on AES, the reader can refer [5].

To conclude, we hope that this thesis provided the reader with some insight on what happens when someone uses WEP to connect to the Internet wirelessly. We also hope that the reader gained some understanding of the kinds of attacks that rendered WEP disadvantageous. As mentioned in Section 5.2, other attacks on WEP are plausible and the reader may be interested in researching and writing about one of those attacks; or even possibly describing how WPA2 operates, its vulnerabilities, and the types of attacks it is subject to.

7.1 Future Study

As mentioned earlier in this chapter, there are multiple types of attacks that can be used when attacking WEP. The reader may be interested in how the FMS attack works when the IV is concatenated to the end of the root key instead of at the beginning, the paper in [28] would be an ideal place to start the research on that topic.

As an alternative, the reader can research some of the work that has been done on RC4 and the proposed changes to RC4 that will provide additional layers for better security. The paper in [16] describes some of these proposed changes.

Other readers might be interested in studying the weaknesses of the KSA part of the RC4 algorithm. The paper in [10] describes the invariance weakness of the KSA in great detail and would be a useful reference for the interested reader.
BIBLIOGRAPHY


APPENDIX A

C++ Code
C++ Code

A.1 First Variation

In this first section of the appendix, the reader can find the C++ code that we used to generate some of the numbers used in the example and counter example of Section 6.5. The reader can use this code freely to check the results mentioned in those sections. As an alternative, the reader can use this code to come generate more examples.

To generate new examples, the reader needs to change the “int Key[]” (on line 7 of the code) to the specific secret key desired. The secret key can be of any length. The $S$ permutation in this code has length 256; and the secret key entered will be transformed to have equal length.

Before beginning the $PRGA$ part of the algorithm, this code will ask for the number of keystream bytes required to be entered. In this manner, the user can decide how many bytes are generated depending on the data to be encrypted.

```
#include <iostream>
int main(int argc, char**argv)
{

int i,j,c,a,b,t,q,z; z=i=j=a=b=q=0;
int permu[256]; /*This will hold the S permutation.*/
int Key[]={3,255,253,52,210,231,155,224,66,75,123,99,100,123,43,213}; /*Change this to the desired secret key*/
int K[256];
c= sizeof(Key) / sizeof(int); /*c is the length of the key entered*/

if ( c == 256 )
{
    printf("The key entered is K = ");
    for (int i = 0; i < c; i++) printf("%d, ", Key[i]);
    printf("\nThe key you entered has length 256.\n");
    for (int i = 0; i < 256; i++) K[i] = Key[i];
}
else if ( c != 256)
{

```
printf("The key entered is K = ");
for (int i = 0; i < c; i++) printf("%d", Key[i]);
printf("\nKey length is not 256. ");
printf("In fact it is %d, c");
printf("The key will be made into a key of length 256.\n");
printf("The new key is K = ");
for (int i = 0; i < 256; i++) K[i]= Key[i%c];
}
for (int i = 0; i < 256; i++) printf("%d", K[i]);
printf("\nInitializing S = ");
for (int i = 0; i < 256; i++) permut[i]=i;
for (int i = 0; i < 256; i++) printf("%d", permut[i]);
/*The KSA part of the algorithm follows*/
printf("\n\nKSA:");
for (int i = 0; i < 256; i++)
{
a=permut[i];
printf("\n %dth step: ni = %d", i, i);
printf(" The previous j=%d", j);
printf(" ... Calculating the new j according to ...");
printf(" j = j + S[i] + K[i] =");
printf("(\%d + \%d + \%d) \mod \%d = \%d", j , permut[i] , K[i]);
printf("((j + permut[i] + K[i])\%256) ");
printf("\nIn other terms, ");
printf("\n we have to Swap S[%d]=%d \, , i, a", i,a);
printf("and S[%d]=", (j + permut[i] + K[i])\%256);
printf("\%d", permut[(j + permut[i] + K[i])\%256]);
j=((j + permut[i] + K[\(i\)])\%256);
b=permut[j];
permut[i]=b; permut[j]=a;
printf("\nTherefore, the new S = \n");
for (int z = 0; z < 256; z++) printf("%d", permut[z]);
printf("\n");
}
/*The PRGA part of the algorithm follows*/
i=j=a=b=t=0;
printf("\n\n How many outputs do you want from PRGA?\n");
scanf("\n %d",&q);
printf("You have entered: %d\n", q);

printf("\n\nPRGA:\n");
for (int k = 0; k < q; k++)
{
    printf("\nStarting with S = ");
    for (int z = 0; z < 256; z++) printf("%d," ,permu[z]);
    printf("\n\n\n");
    i=i+1;
    printf("i = %d ",i);
    b=permu[i];
    printf("So S[%d] = S[%d] = %d \n",i,b);
    j=j+permu[i];
    j=j%256;
    printf("j = %d ",j);
    a=permu[j];
    printf("So S[%d] = S[%d] = %d \n",j,a);
    printf("So we swap S[%d] and S[%d],i,j);
    printf(" to get the new S[%d] = %d and S[%d] = %d",i,a,j,b);
    permu[i]=a;
    permu[j]=b;
    printf(" ... and so ... S = ");
    for (int t = 0; t < 256; t++) {printf("%d," ,permu[t]);}
    printf("\n");
    t=a+b;
    t=t%256;
    printf("\n Then t = S[%d] + S[%d] = %d\n",i,j,t);
    printf(" and S[t] = %d\n",permu[t]);
    printf(" In other terms,\n");
    printf("the %dth keystream byte is S[t] = %d\n",k,permu[t]);
    printf("___________________________\n");
}
return 0;
system("PAUSE");
}

A.2 Second Variation

In this section of the appendix, we present the reader with a less complicated C++ code. This code does not expand the secret key entered. The reader can change the “int Key[]” to the desired secret key. The secret key can be of any length, and this also has to be entered. This code will determine the size the entered key and will set the $S$ permutation to have equal length to the key. Finally, this code will generate one keystream byte only.

```cpp
#include <iostream>
int main(int argc, char**argv)
{
    /*The KSA part of the algorithm follows*/
    int i,j,c,a,b,t; i=j=a=b=0;

    int permu[256];/*This will hold the S permutation*/
    int K[]={1,5,2,3,7};/*Enter your key here*/
    c= sizeof(K) / sizeof(int);

    for (int i = 0; i < c; i++){/*printf("i = %d ",i);*/ permu[i]=i;}
    printf("N = %d
",c);
    printf("size of K = %d\n",c);
    printf("you have entered: K =");
    for (int i = 0; i < c; i++) {printf(" %d ",K[i]);}
    printf("and S =");
    for (int i = 0; i < c; i++) {printf(" %d ",permu[i]);}printf("\n");
    printf("KSA:");
    for (int i = 0; i < c; i++)
    {
        a=permu[i];
        printf("\n %dth step; i = %d",i,i);
        /*printf(" before j=%d",j);*/
        printf(" ... (Calculating j) ... j = j + S[i] + K[i]
= (%d + %d + %d) (mod %d) = %d",j , permu[i] , K[i],
c,((j + permu[i] + K[i])%c ));
```

```
printf(" ... So Swap S[%d] and S[%d]"
, i,(((j + permu[i] + K[i])%c) ));
j=(((j + permu[i] + K[(i)])%c);
b=permu[j];
permu[i]=b; permu[j]=a;
printf(" ... and so");
printf("\n ... S = ");
for (int t = 0; t < c; t++) {printf(" %d ",permu[t]);}
printf("\n");
}

/*The PRGA part of the algorithm follows*/
i=j=a=b=t=0;
printf("\n\nPRGA:\n");
for (int k = 0; k < 1; k++)//change i<1 to i<# if you
want more keystreams bytes.
{
    printf("\nStarting with S = ");
    for (int t = 0; t < c; t++) {printf(" %d ",permu[t]);}
    printf("\n");
    i=i+1;
    printf("i = %d ",i);
    b=permu[i];
    printf("So S[i] = S[%d] = %d \n",i,b);
    j=j+permu[i];
    j=j%c;
    printf("j = %d ",j);
    a=permu[j];
    printf("So S[j] = S[%d] = %d \n",j,a);
    printf("So we swap S[%d] and S[%d] to get the new S[%d]
    = %d and S[%d] = %d",i,j,i,a,j,b);
    permu[i]=a;
    permu[j]=b;
    printf(" ... and so ... S = ");
    for (int t = 0; t < c; t++) {printf(" %d ",permu[t]);}
    printf("\n");
t = a + b;
t = t \% c;
printf("\n Then t = S[%d] + S[%d] = %d", i, j, t);
printf(" and S[t] = %d\n", permu[t]);
printf(" In other terms, the %dth keystream byte is
S[t] = %d\n", k, permu[t]);
printf("____________________________\n");
}
return 0;
system("PAUSE");
}