KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

A Security Analysis of the WPA-TKIP and TLS Security Protocols

Mathy Vanhoef

Supervisor:
Prof. dr. ir. F. Piessens

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering Science: Computer Science

July 2016

A Security Analysis of the WPA-TKIP and TLS Security Protocols

Mathy VANHOEF

Examination committee:
Prof. dr. ir. P. Van Houtte, chair
Prof. dr. ir. F. Piessens, supervisor
Prof. dr. ir. C. Huygens
Prof. dr. D. Hughes
Prof. dr. ir. B. Preneel
Prof. dr. K. Paterson
(Royal Holloway, University of London)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering Science: Computer Science

July 2016

Preface

When I started my PhD four years ago, I thought people were exaggerating when they said that doing a PhD was a unique and special experience. Turns out I was wrong: it is indeed quite the journey! It was a roller coaster of exciting and sometimes no so exciting research results, late-night deadline races, moments of doubt, enriching and rewarding travels, grueling dilemmas, facing your own limitations,... you get the idea. This journey would not have been possible without the help of several people, and with the risk of forgetting some, I would like to thank them here.

First and for all, I want to thank my supervisor Frank Piessens. You gave me the freedom to work on my own ideas, for which I am forever grateful. Your quick understanding of new ideas, insightful comments, and overall guidance has truly been inspiring.

I would also like to thank the members of my jury, Frank Piessens, Christophe Huygens, Danny Hughes, Bart Preneel, and Kenny Paterson, for taking the time to read my dissertation, and providing me with helpful comments in preparation of this final version. And Paul Van Houtte, thank you for chairing the jury.

Special thanks also goes to my office mates. Frédéric Vogels, you always made sure people felt welcome and enjoyed their day. Since I was new to Leuven, and the department, this was greatly appreciated. Thanks for all the moelleux, and for all the pedantic but entertaining discussions. Raoul Strackx, you have been a great colleague and friend. Thanks for always being willing to help out with things when needed. And of course, thanks for enduring all the witty remarks over the last few years, you have survived them well. Jesper Cockx, thanks for making the office a fun place, and for introducing us to D&D. May there be many more evil spiders and monsters to kill.

A big thank you also goes to our lunch group: Pieter Agten, Sven Akkermans, Jesper Cockx, Milica Milutinović, Andreas Nuyts, Marco Patrignani, Rula Sayaf, Raoul Strackx, Jan Tobias Mühlberg, Jo Van Bulck, Gitte Vanwinckelen,

Frédéric Vogels, and Neline van Ginkel. Thanks to all of you, lunch was always something to look forward to. I also enjoyed all our after work get-togethers, we should keep organizing them!

I would also like to thank all the people I have met, or worked with, during my PhD. First, Kenny Paterson, thanks for your enthusiastic reaction to my initial but vague ideas about attacking RC4. Your positive reaction made me research RC4 in more detail! Tom Van Goethem, our recent collaborations, including late-night deadline work, clearly paid off. I hope we can continue this great work in the future. I also appreciated working with, and learning from, my fellow teaching assistants and lecturers: Yolande Berbers, Jasper Bogaerts, Philip Dutré, Wouter Franken, Roald Frederickx, Job Noorman, Steven Op de beeck, Willem Penninckx, Jan Smans, Eric Steegmans, and Thomas Winant. And of course I also enjoyed meeting all my other colleagues, who helped made every day at the department a pleasant one: Wilfried Danëls, Willem De Groef, Philippe De Ryck, Maarten Decat, Lieven Desmet, Dominique Devriese, Bart Jacobs, Wouter Joosen, Bert Lagaisse, Jef Maerien, Jonathan Merlevede, Nick Nikiforakis, Davy Preuveneers, Jose Proenca, Bob Reynders, Zubair Rafique, Jan Spooren, Klaas Thoelen, Gijs Vanspauwen, Thomas Vissers, and Rinde van Lon. Also a thank you to Bram Bonne, Mariano Di Martino, Pieter Robyns, and Dieter Vandenbroeck, for the discussions we had over the years.

And of course there is Hacknamstyle, our capture the flag team. Steven Van Acker, thanks for blowing new life into this group, for all the discussions we had, and for showing me how to run experiments on all computers of the department. Tom Van Goethem and Neline van Ginkel, keep up the good work you are doing for Hacknamstyle! Without you the group just would not be the same. It has also been great to meet all the people who regularly participated in the workshops or CTFs: Pieter Agten, Jeroen Beckers, Mathias Bynens, Dieter Castel, Xavier Castermans, Anthony Clays, Thomas De Backer, Kevin Holvoet, Sophie Marien, Michiel Meersmans, Arne Swinnen, Bert Van de Poel, Jo Van Bulck, Tim Van den Eynde, Kobe Vrancken, Koen Yskout, and Bart van der Plancken. Thank you all for joining our activities and creating a fun atmosphere.

And to all the people of the secretariat, administration, and business office, thank you for all the help and for keeping our paper work to a minimum: Katrien Janssens, Fred Jonker, Anne-Sophie Putseys, Esther Renson, Ghita Saevels, Marleen Somers, Karen Spruyt, and Annick Vandijck.

This work also would not have been possible without the people that manage the computer infrastructure at our department. Even when I was running experiments on nearly every computer, and was heating up cold classrooms during the winter, at least according to students, they allowed me to keep running my computations. Thank you!

I would also like to thank the Research Foundation - Flanders (FWO) for granting me a PhD fellowship, and the Research Fund KU Leuven for their financial support.

Als laatste, maar daarom zeker niet minder belangrijk, wil ik mijn ouders en familie bedanken voor alle steun doorheen de jaren. Zonder jullie was dit werk niet mogelijk! Aan mijn twee broers Peter en Erik, bedankt om mij van jongs af aan al een kritische ingesteldheid bij te brengen, en mij te laten *prullen* met jullie computers. Het heeft duidelijk zijn effect gehad!

Thank you all!

— Mathy Vanhoef
Leuven, July 2016

Abstract

This dissertation analyzes the security of popular network protocols. First we investigate the Wi-Fi Protected Access Temporal Key Integrity Protocol (WPA-TKIP), and then we study the security of the RC4 stream cipher in both WPA-TKIP and the Transport Layer Security (TLS) protocol. We focus on these protocols because of their popularity. In particular, around November 2012, WPA-TKIP was used by two-thirds of encrypted Wi-Fi networks, and it is currently still used by more than half of all encrypted networks. Similarly, around 2013, RC4 was used in half of all TLS connections. Finally, with as goal to implement reliable proof-of-concepts for some of our attacks against WPA-TKIP, we also study physical layer security aspects of Wi-Fi.

In the first part of this dissertation we focus on WPA-TKIP when used to protect unicast Wi-Fi traffic. Here we demonstrate how fragmentation of Wi-Fi frames can be used to inject an arbitrary number of packets, and we show how this attack can be applied in practice by performing a portscan on any client connected to the network. Then we propose a technique to decrypt arbitrary packets sent towards a client. Our technique first resets the internal state of the Michael algorithm, and abuses this to make victims forward packets to a server under control of the adversary, effectively decrypting the packets. We also present a novel Denial of Service (DoS) attack that requires the injection of only two frames every minute. Additionally, we discover that several network cards use flawed and insecure implementations of WPA-TKIP.

In the second part of the dissertation, our goal is to attack WPA-TKIP when used to protect broadcast and multicast traffic, i.e., group traffic. This is important since, even in 2016, more than half of all encrypted Wi-Fi networks still protect group traffic using WPA-TKIP. To carry out our attack in a general setting, we must be able to reliably block certain packets from arriving at their destination, preferably using cheap commodity Wi-Fi devices. Hence we first study low-layer aspects of the Wi-Fi protocol. Surprisingly, we found that commodity devices allow us to violate several assumptions made by the Wi-Fi

protocol. We show this enables us to implement a constant and selective jammer using commodity Wi-Fi devices. Although the selective jammer can block a large percentage of packets from arriving at their destination, we found that an even more effective method is to block packets by obtaining a channel-based man-in-the-middle (MitM) position. In such a position, packets are blocked by not forwarding them. Finally, we demonstrate that our MitM position allows us to attack WPA-TKIP, when used as a group cipher, within only 7 minutes.

In the last part of the dissertation we attack RC4 in both WPA-TKIP and TLS. First we search for new biases in the RC4 keystream, in hope they might be useful to improve our attacks. We empirically search for them using statistical hypothesis tests. This reveals many new biases in the initial keystream bytes, as well as several new long-term biases. Then we design algorithms that are capable of using multiple types of biases, in order to recover a repeatedly encrypted secret. These algorithms return a list of plaintext candidates in decreasing likelihood, and are applied to attack WPA-TKIP and TLS. For the WPA-TKIP scenario we first introduce a method to generate a large number of identical packets. We decrypt this packet by generating its plaintext candidate list, and use redundant packet structure to prune bad candidates. From the decrypted packet we derive the WPA-TKIP MIC key, which can be used to inject and decrypt packets. In practice the attack can be executed within an hour. In the attack against TLS, we show how to decrypt a secure HTTP cookie with a high success rate, by capturing roughly one billion ciphertexts. This is done by injecting known data around the cookie, abusing this using Mantin's $AB\mathcal{S}AB$ bias, and brute-forcing the cookie by traversing the plaintext candidates. Using our traffic generation technique, we are able to execute the attack, and decrypt the cookie, within merely 75 hours.

Beknopte samenvatting

In deze doctoraatstekst analyseren we de veiligheid van populaire netwerkprotocollen. Eerst onderzoeken we het Protected Access Temporal Key Integrity Protocol (WPA-TKIP), en daarna bestuderen we de veiligheid van het RC4 stroomcijfer in zowel WPA-TKIP als het Transport Layer Security (TLS) protocol. We focussen op deze protocollen vanwege hun populariteit. Meer bepaald, in november 2013 werd WPA-TKIP gebruikt door twee derde van alle geëncrypteerde Wi-Fi netwerken, en momenteel wordt het nog steeds door meer dan de helft van alle geëncrypteerde netwerken gebruikt. En rond 2013 werd RC4 door ongeveer de helft van alle TLS connecties gebruikt. Tenslotte onderzoeken we ook de veilig van de fysieke laag van het Wi-Fi protocol, zodat we algemene en betrouwbare aanvallen tegen WPA-TKIP kunnen implementeren.

In het eerste deel van de doctoraatstekst bestuderen we WPA-TKIP wanneer het gebruikt wordt om unicast netwerkverkeer te beschermen. We demonstreren hoe fragmentatie van Wi-Fi pakketten gebruikt kan worden om een willekeurig aantal pakketten te versturen, en we tonen aan hoe dit in de praktijk kan worden gebruikt om een portscan uit te voeren op eender welke computer op het netwerk. Vervolgens stellen we een techniek voor om willekeurige pakketten, die verzonden zijn door een access point, te decrypteren. Deze techniek reset eerste de interne staat van het Michael algoritme, en bouwt hierop verder om ervoor de zorgen dat slachtoffers pakketten doorsturen naar een computer onder controle van de aanvaller. Hierdoor leert de aanvaller de inhoud van de pakketten. We tonen ook aan dat er een denial-of-service aanval bestaat tegen WPA-TKIP waarbij de aanvaller slechts twee pakketten elke minuut moet versturen. Bovendien laten we zien dat verscheidene netwerkkaarten onbetrouwbare en onveilige implementaties van WPA-TKIP gebruiken.

In het tweede deel van deze tekst onderzoeken we de veiligheid van WPA-TKIP wanneer het gebruikt wordt om broadcast en multicast verkeer te beschermen. Dit is belangrijk aangezien meer dan de helft van alle geëncrypteerde Wi-Fi netwerken nog altijd WPA-TKIP gebruikt om dit verkeer te beschermen, zelfs in

2016. On de aanval uit te voeren moet een aanvaller specifieke Wi-Fi pakketten kunnen blokkeren, en dit liefst met goedkope toestellen. Om dit te bereiken bestuderen we eerste de fysieke laag van het Wi-Fi protocol. Verrassend genoeg ontdekten we dat veelgebruikte Wi-Fi toestellen kunnen gebruikt worden als een constante of selectieve stoorzender. Hoewel de selectieve stoorzender de meeste Wi-Fi pakketten kan blokkeren, en deze pakketten dus niet door de andere apparaten correct ontvangen zal worden, is een nog betere manier om een kanaal-gebaseerde man-in-the-middle (MitM) aanval te gebruiken. In deze MitM aanval kunnen pakketten geblokkeerd worden door ze niet door te sturen naar hun uiteindelijke bestemming. Door hiervan gebruikt te maken, tonen we aan dat als WPA-TKIP wordt gebruikt om multicast en broadcast pakketten te beschermen, het kan worden aangevallen binnen 7 minuten.

Tenslotte vallen we het RC4 stroomcijfer aan in zowel WPA-TKIP als TLS. Eerst zoeken we naar niet-uniforme bytes in de sleutelstroom van RC4. We gebruiken hiervoor een empirische techniek die gebruik maakt van statistische toetsen. Met behulp van deze techniek vonden we verscheidene, nog niet gekende, niet-uniforme bytes in de sleutelstroom van RC4. Vervolgens ontwerpen we algoritmes die verschillende niet-uniforme sleutelstroombytes combineren om een herhaaldelijk geëncrypteerd geheim te achterhalen. Deze algoritmes berekenen een lijst van klaartekst kandidaten in aflopende volgorde van waarschijnlijkheid, en worden toegepast om WPA-TKIP en TLS aan te vallen. Voor de aanval tegen WPA-TKIP genereren we eerst een groot aantal identieke pakketten. Dit pakket wordt gedecrypteerd door de klaartekst kandidaten te berekenen, en ongeldige kandidaten te filteren op basis van de structuur van WPA-TKIP pakketten. Eens we de klaartekst van het pakket weten, kunnen we de WPA-TKIP MIC sleutel berekenen, waarmee we pakketten kunnen decrypteren en versturen. De aanval kan binnen een uur uitgevoerd worden. In de aanval tegen TLS tonen we aan dat een aanvaller ongeveer één miljard sleutelteksten moet onderscheppen om een veilige HTTP cookie te kunnen decrypteren. Eerst zorgen we dat er gekende klaartekst rondom het HTTP cookie staat, zodat we het niet-uniforme $AB\mathcal{S}AB$ patroon in de sleutelteksbytes van RC4 kunnen gebruiken om een lijst van klaartekst kandidaten te genereren. Elk van deze kandidaten wordt actief getest totdat de juiste klaartekst is gevonden. Met behulp van een techniek om de sleutelteksten snel te genereren, kunnen we deze aanval uitvoeren in slechts 75 uren.

Contents

List of Figures

List of Tables

List of Abbreviations

AC	Access Class.
AES	Advanced Encryption Standard.
AIFS	Arbitration Interframe Space.
AIFSN	Arbitration Interframe Space Number.
AP	Access Point.
API	Application Programming Interface.
ARP	Address Resolution Protocol.
CBC	Cipher Block Chaining.
CCA	Clear Channel Assessment.
CCM	CTR with CBC-MAC.
CCMP	CTR with CBC-MAC Protocol.
CRC	Cyclic Redundancy Code.
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance.
CTS	Clear To Send.
CW	Contention Window.
DCF	Distributed Coordination Function.
DMA	Direct Memory Access.
DNS	Domain Name System.

DoS	Denial of Service.
EAPOL	Extensible Authentication Protocol Over LANs.
EDCA	Enhanced Distributed Channel Access.
FCS	Frame Check Sequence.
FIPS	Federal Information Processing Standards.
FM	Fluhrer-McGrew.
FMS	Fluhrer-Mantin-Shamir.
GTK	Group Temporal Key.
HMAC	Keyed-Hash Message Authentication Code.
HMM	Hidden Markov Model.
HTTP	Hypertext Transfer Protocol.
HTTPS	HyperText Transfer Protocol Secure.
ICMP	Internet Control Message Protocol.
ICV	Integrity Check Value.
IEEE	Institute of Electrical and Electronics Engineers.
IFS	Interframe Space.
IMAP	Internet Message Access Protocol.
IP	Internet Protocol.
IV	Initialization Vector.
KSA	Key Scheduling Algorithm.
LAN	Local Area Network.
LEAP	Lightweight Extensible Authentication Protocol.
LLC	Logical Link Control.
MAC	Medium Access Control.
MIC	Message Integrity Code.
MitM	Man-in-the-Middle.
MPDU	MAC Protocol Data Unit.

MSDU MAC Service Data Unit.

PCI Peripheral Component Interconnect.

PCIe PCI Express.

PLCP Physical Layer Convergence Procedure.

PRGA Pseudo Random Generation Algorithm.

PTK Pairwise Transient Key.

QoS Quality of Service.

RAM Random-Access Memory.

RFC Request For Comment.

RTS Request To Send.

SIFS Short Interframe Space.

SMTP Simple Mail Transfer Protocol.

SNAP Sub-Network Access Protocol.

SSID Service Set Identifier.

SSL Secure Socket Layer.

TCP Transport Control Protocol.

TID Traffic Identifier.

TK Temporal Key.

TKIP Temporal Key Integrity Protocol.

TLS Transport Layer Security.

TSC TKIP Sequence Counter.

TTL Time To Live.

UDP Unreliable Datagram Protocol.

USB Universal Serial Bus.

USRP Universal Software Radio Peripheral.

WEP Wired Equivalent Privacy.

WPA Wi-Fi Protected Access.

Chapter 1

Introduction

> "I've hacked into the Covenant battlenet. They're actually broadcasting tactical data on unencrypted channels. We should show them who they're dealing with."
>
> — *Cortana, Halo 1*

The internet has dramatically revolutionized society and our everyday lives. Preceded by the invention of the telegraph, telephone, radio, and satellite, it proved to be an unprecedented medium for sharing and disseminating information [90]. An essential ingredient for this success was that it enabled fast and inexpensive real-time interactions between people, computers, or both, independent of their geographical location. In turn, such interactions were made possible by real-time connections between computers, and these connections can be seen as the foundation of the internet. In the early days of the internet, when it was still called the advanced research projects agency network (ARPANET), all computers in this network were trusted. Consequently, all connections were implicitly trusted as well, meaning there was no need to protect transmitted messages, nor to verify received ones. However, as the internet evolved, and businesses were eager to use it for commercial ends, there was a sudden need for secure communications. In response, Netscape Communications proposed the Secure Sockets Layer (SSL) protocol in 1994 [167]. Gradually, it evolved to become the Transport Layer Security (TLS) protocol, which is currently a de facto standard for securing internet traffic.

However, the evolution of Netscape's SSL protocol was not a smooth journey. Along the way, several serious attacks against it were discovered, prompting

the need for newer and more secure versions. In fact, the initial version of SSL was never publicly released due to major flaws in its design. The second version, denoted by SSLv2, did not fare much better. It used the same key for encryption and message integrity and did not securely terminate sessions [169]. Additionally, it did not defend against downgrade attacks: a man-in-the-middle could modify handshake messages, tricking the client into picking a weaker cipher suite than it would normally choose [169]. These attacks led to the development of SSLv3, which forms the basis of the current TLS protocols.

This ad hoc approach to security, where new versions of a protocol are primarily introduced or adopted to defend against known attacks, is a reoccurring phenomenon. In certain cases, it even takes everyday attacks to finally push people into developing and adopting more secure protocols. For example, Ylönen only started the development of Secure Shell (SSH) after his university network was the victim of password sniffing attacks [14]. Before the release of SSH, people were using Remote Shell (RSH) without any form of encryption or protection, even though they realized this was far from ideal [159]. Another relevant example is the DNS protocol. Maintaining its security has been a constant challenge, where high-profile attacks were followed with ad hoc countermeasures in an attempt to safeguard its security [4]. Yet another example is the contactless Mifare Classic smartcard. After its release several serious attacks against it were discovered, resulting in the development of backwards-compatible countermeasures. However, these solutions still relied on the flawed cryptographic algorithm of the Mifare classic card, and recently it was shown that attacks remain possible, even if all the ad hoc countermeasures are implemented [106].

Summarized, we observe that the evolution of secure network protocols has been predominantly influenced by the discovery and threat of (both realistic and theoretic) attacks against their design or implementation. This dissertation explores and analyzes new weaknesses introduced during this reactive and ad hoc approach to security, proposes novel techniques to exploit known flaws, and demonstrates the impact of resulting attacks in real-world scenarios. In particular we will first study link-layer protocols, where we focus on Wi-Fi networks and the WPA-TKIP protocol. We then extend our focus to include network and transport layer protocols, and analyze the usage and security of RC4 in both TLS and WPA-TKIP. These protocols also serve as good examples that, commonly, sufficiently realistic and convincing attacks are necessary to force the adoption of newer and more secure versions of a protocol.

1.1 Background

We start by introducing basic cryptographic building blocks that are used in the protocols we will investigate. Additionally, we define common terms used in this dissertation.

Cryptographic ciphers can be classified as being either symmetric of asymmetric. Symmetric ciphers use the same key for both encryption and decryption. In contrast, asymmetric ciphers use two different keys. One key is kept secret and used for decryption, and is called the private key. The other key is called the public key, and is used for encryption. Generally asymmetric ciphers are used in an initial phase to negotiate or generate session keys. These session keys are then used by symmetric ciphers to encrypt and protect data. This is done to reduce the heavier computational cost of asymmetric ciphers.

In this dissertation we focus on symmetric encryption schemes, and hence we will further discuss these. In particular, symmetric ciphers can be divided into stream and block ciphers, and we will briefly introduce both.

1.1.1 Stream ciphers

Stream ciphers produce a large chunk of secret, random looking output called the keystream. Encryption is performed by combining the plaintext with the generated keystream using an exclusive OR (XOR) operation. The output of a strong stream cipher should be indistinguishable from a random stream of bytes. This leads to a common attack model for stream ciphers: an adversary should not be able to (efficiently) tell the difference between completely random data, or output from the stream cipher. An algorithm which tries to detect this difference is called a distinguisher. The distinguisher is given either truly random data, or output of the stream cipher, and it attempts to determine which of the two inputs it was given. It can try to do this by searching for biases in the given data stream. There are two types of distinguishers [64]:

Weak distinguisher A weak distinguisher is given a single output stream, which is either output of the stream cipher, or truly random data.

Strong distinguisher A strong distinguisher is given a black box which either generates truly random data, or contains an implementation of the stream cipher which generates keystream. The black box can be restarted with new random keys polynomially many times.

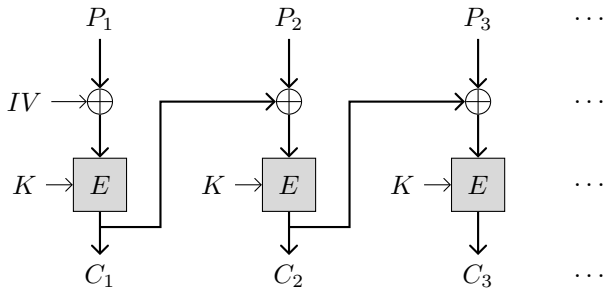

Figure 1.1: Cipher Block Chaining (CBC) mode.

While it was shown that theoretically these distinguishers can detect the same types of biases [64, §3.2.3], in practice there can be a major difference in the type of biases they can (easily) detect [100]. This is beautifully demonstrated by the strong bias towards zero at position two of the RC4 keystream. After the description of RC4 was leaked, it took 8 years until this bias was discovered by Mantin and Shamir. They claim the reason this bias was not discovered earlier was because previous works analyzed RC4 using weak distinguishers, instead of strong ones [100].

1.1.2 Block ciphers

Block ciphers encrypt data blocks of a fixed size. The size of a block is determined by the specific cipher being used. One of the most widely-used block ciphers is the Advanced Encryption Standard (AES), also known as Rijndael [38]. Because the length of the data being encrypted often does not equal the length of the block cipher, a mode of operation needs to be defined in order to handle data of arbitrary length. Here we will present two such methods, namely Cipher Block Chaining (CBC) and counter (CTR) mode.

One of the first proposed modes of operation for block ciphers was CBC [47]. In CBC mode, the plaintext is first padded to a multiple of the block size. The padded plaintext is then divided into blocks $P_1, P_2, \ldots, P_n$, and along with an Initialization Vector (IV) and a key K, it is handed over to CBC for encryption. The output is a list of ciphertext blocks C_1, $C_2, \ldots, C_n$, whose generation is illustrated in Fig. 1.1.

Another widely used method is Counter mode with CBC-MAC (CCM) [190], which turns a block cipher into a stream cipher. This is done by using the block cipher to encrypt the concatenation of a nonce and a counter, and treating the

output as keystream. The counter is increased for every block of generated keystream. Here the nonce is similar to an IV, which must be unique for every message being encrypted. In addition to encryption, CCM also provides message authentication using CBC-MAC, which is a variant of CBC mode that calculates a message authentication code.

1.2 The Evolution of Network Protocols

Protocols generally undergo several iterations before reaching a sufficient level of security [7, 12, 134, 169]. Each iteration improves the security it provides, and this is commonly accompanied with the implementation of countermeasures to mitigate attacks against older versions. In the next section we argue that the adoption of these newer versions generally has been a slow process, in the sense that people only adopt new versions when under the threat of convincing attacks. Therefore, when researching the security of real-world systems, it is essential to first determine which (version of) protocols are being used. After this, one can analyze the security of protocols actually in use, instead of merely focussing on the latest version of a particular protocol. With this in mind, we start by briefly outlining the evolution of both Wi-Fi and TLS.

1.2.1 Wi-Fi

The 802.11 protocol, commonly known as Wi-Fi, initially provided Wired Equivalent Privacy (WEP) in order to protect traffic [192, §11.2.2]. For encryption WEP uses RC4, and data authentication is implemented using an encrypted Cyclic Redundancy Check (CRC). Researchers quickly highlighted major flaws in the design of WEP [109]. The most devastating attack was found by Fluhrer, Mantin, and Shamir in 2001 (the FMS attack [57]). They showed that capturing sufficiently many packets allowed an adversary to recover the private key. Initially the industry disregarded this attack as "minor and sophisticated" [109]. One common countermeasure was to use key management protocols to frequently renew the private WEP key [27]. In 2006 more than 75% of encrypted networks still used WEP. This prompted Bittau et al. [27] to improve existing attacks, and they came up with an attack requiring less than a minute to allow an adversary to inject and decrypt packets. As a result, the frequent re-keying countermeasure was shown to be inadequate, further illustrating that WEP should be abandoned.

To provide an alternative to the broken WEP protocol, the IEEE started with designing both a short-term and long-term solution, in what would become

the 802.11i amendment to the 802.11 standard [7]. Their long-term solution is based on the AES cipher [38] operating in CCM mode (counter mode with CBC-MAC [190]). Commonly this protocol is called (AES-)CCMP. Unfortunately, because many wireless chips implement their cryptographic algorithms in hardware, old devices would not be able to support AES-CCMP. For these older devices a short-term solution called the Temporal Key Integrity Protocol (TKIP) was designed. It was designed so that old WEP-capable hardware could support TKIP through firmware upgrades. It took until 2004 before both protocols were ratified by the IEEE in the 802.11i amendment.

In 2003 the Wi-Fi Alliance, frustrated with the slow standardization process of the IEEE, already started certifying devices based on a draft version of the 802.11i amendment. They called this certification program Wi-Fi Protected Access (WPA). The program required that WPA certified devices supported TKIP, and allowed, but did not mandate, support of AES-CCMP. This lead to the common misuse of the term WPA: it is not synonymous with TKIP. In 2006 the WPA2 certification program was created, and it required support for AES-CCMP while optionally allowing TKIP. Unfortunately this led to another common misunderstanding: in practice a WPA2 network will still support TKIP, unless explicitly configured to only allow AES-CCMP. In this dissertation we will use WPA-TKIP to explicitly refer to TKIP, and use the term WPA to refer to Wi-Fi encryption in general.

1.2.2 Transport Layer Security (TLS)

Version 1.0 of TLS was published in 1999 and was heavily based on SSLv3 [41]. One important difference with SSLv3 is that TLSv1.0 defined the content of padding bytes when using a CBC-mode cipher suite. This made it immune against the POODLE attack [115], though some implementations of TLSv1.0 were still vulnerable because they forgot to verify the content of these received padding bytes [88]. Additionally, TLSv1.0 explicitly addressed the Bleichenbacher [28] padding oracle attack against RSA, by unifying certain error messages, and closing timing based side channels. In contrast, SSL did not anticipate the Bleichenbacher attack, and implementations had to undergo ad hoc patches in order to mitigate the attack.

Sadly, TLSv1.0 still suffered from several design flaws [158]. One example is the BEAST attack [46], which exploited known weaknesses of how SSL and TLSv1.0 implement CBC-mode using chained initialization vectors [11]. Another weakness is that TLSv1.0 was vulnerable to side channel attacks that abused unauthenticated padding in CBC-mode encryption [33, 188]. Implementations were patched by closing these side channels [114].

The above issues were addressed in the release of version 1.1 of TLS [42]. While the previous padding side channels attacks against CBC-mode encryption were now addressed in the standard, there was still a small but exploitable timing side channel left [3, 6]. Vulnerable implementations had to be patched to remove this timing side channel. Version 1.2 was published in 2008 and mainly added support for more modern cipher suites, such as the usage of SHA-256, and support for authenticated encryption ciphers (e.g. AES-GCM) [43]. These latest versions of TLS were found to be vulnerable against client impersonation attacks, which rely on a clever combination of the various types of handshakes that TLS supports [24]. The draft specification of TLSv1.3 addresses this attack, along with several other improvements [44].

1.3 Overview of Selected Attacks on TLS and WPA

This section provides an overview of recent attacks on SSL/TLS and WPA. We first cover attacks that abused (known) weaknesses in older protocols, and that had a significant influence on the adoption of new protocols. These attacks highlight that at times sufficiently practical and detrimental attacks were needed in order to motivate people in disabling old and weak protocols. Finally, we go over other noteworthy attacks, to provide a broader context for the results presented in this dissertation.

1.3.1 The power and necessity of convincing attacks

Five years ago Facebook did not offer HTTPS by default to all its users [105]. Similarly, Google and Twitter also did not offer HTTPS by default [149, 170]. Instead, they only used HTTPS to protect the transmission of passwords when logging in. Once logged in, they fell back to using insecure connections, allowing an adversary to trivially intercept session cookies and impersonate the user. Moreover, this practice facilitated SSL/TLS stripping attacks [103], generally still allowing an active attacker to intercept a user's password. One reason why SSL/TLS was not yet widely adopted was due to the performance cost of encryption, and in particular key agreement. However, around 2010 companies such as Google showed that the overhead of SSL/TLS is no longer a concern [87]. For example, they were able to enable HTTPS by default on Gmail without requiring additional machines or hardware. Combined with the increased public awareness that accounts can get hijacked on websites that do not always use HTTPS [92], we believe this motivated companies to finally adopt HTTPS. The increased public awareness was caused by FireSheep [48, 92]: an easy to use

Firefox extension that monitors nearby Wi-Fi networks, allowing one to trivially intercept session cookies to hijack accounts on widely used networks [30].

In 2006 Bard introduced an attack against SSL and TLSv1.0 that abused the predictability of Initialization Vectors (IVs) used in Cipher Block Chaining (CBC) modes [11]. Essentially, SSL and TLSv1.0 do not generate a new IV for every message, but reuse the previous ciphertext block as an IV, making it possible to mount an attack that can recover low entropy information. Bard hoped his attack would speed-up the adoption of TLSv1.1 or higher. Unfortunately, it did not significantly increase adoption rates. However, in 2001, Duong and Rizzo implemented a more practical variant of Bard's attack which they called BEAST [46]. Perhaps the most influential and novel aspect of BEAST is the attack scenario: they assume an adversary can run malicious code inside the victim's browser, *and* monitor encrypted network traffic. While this attack received a large amount of media attention, and hence made people more aware of the issues with TLSv1.0, the adoption of TLSv1.1 or higher still remained relatively low. As a result, to mitigate the BEAST attack against SSLv3 or TLSv1.0, the recommendation was to use RC4 [142].

After TLSv1.0 was found to be vulnerable to side channels attacks that relied on the presence of unauthenticated padding in CBC-mode cipher suites [33, 188], later versions of TLS explicitly closed these side-channels [42, 43]. In particular, versions 1.1 and 1.2 contain the following countermeasures (emphasis mine):

> [..] compute the MAC even if the padding is incorrect, and only then reject the packet. For instance, if the pad appears to be incorrect, the implementation might assume a zero-length pad and then compute the MAC. *This leaves a small timing channel*, since MAC performance depends to some extent on the size of the data fragment, *but it is not believed to be large enough to be exploitable*, due to the large block size of existing MACs and the small size of the timing signal.

Strangely, they did not address all timing-based side channel attacks, believing these would not be exploitable in practice. It took a successful attack before people realized and accepted that this timing difference *is* exploitable [6]. Tedious patches were required to properly address this timing side channel [3].

Many SSL and TLS implementation include several old and weak cipher suites, including export ciphers suites that are deliberately weakened to comply with old export regulations on cryptography. In attacks such as FREAK [23], Logjam [1], and SLOTH [25], it was shown that supporting these weak ciphers may introduce critical vulnerabilities. As a result, it is now recommended to disable support of these weak ciphers suites [2, 78].

In 2016 Aviram et al. found that 17% of all HTTPS servers still support the outdated SSLv2 protocol [10], in spite of it being prohibited in 2011 [169]. They showed that these SSLv2 servers can be used as an oracle to decrypt a (modern) RSA-based TLS handshake, under the condition that the older SSLv2 server uses the same private key as the modern TLS server. Their attack is based on a variant of the Bleichenbacher RSA padding oracle attack [28]. Due to this new attack, people are more serious about disabling support of SSLv2 [45, 76, 140].

In 2003 it was demonstrated that LEAP, an old authentication mechanism that can be used in enterprise WPA networks, was vulnerable to dictionary attacks [194]. Moreover, around this time the general advice was to run these old authentication mechanism inside a TLS tunnel [9]. However, this practice was never adopted for LEAP. While Cisco warned to stop using LEAP [40], Apple by default continued to support this outdated authentication mechanism. Only in 2014, after it was shown that LEAP was also vulnerable against man-in-the-middle and impersonation attacks, did Apple disable it [145].

1.3.2 Other prominent attacks

Two other attacks that received significant attention from the security community were CRIME and BREACH [63, 144]. The CRIME attack abuses compression at the TLS level to recover cookies. In contrast, the BREACH attack abuses compression at the HTTP level to recover secret information contained in web pages (e.g., cross site request forgery tokens). They can be prevented by disabling compression at the TLS or HTTP layer, respectively.

One of the more noteworthy attacks against enterprise WPA networks is the one by Cassola et al. [34]. These networks typically require a username and password for authentication. Cassola et al. show how to bypass server identity checks in order to obtain a man-in-the-middle position, to then intercept users' credentials. In enterprise WPA networks, clients normally protect against man-in-the-middle attack by pinning (or preconfiguring) the public key of the authentication server. This check can be bypassed by advertising a network with a visually similar SSID. For example, if the target network is called "`Enterprise`", they create a rogue AP with the SSID "`Enterprise`␣". In the interface of most operating systems, both SSIDs are visually identical. When the victim connects to the rogue network, no warning is shown since there is no expected public key configured for this malicious SSID. Nearly all victims proceed by (re-)entering their credentials [34], effectively handing them over to the adversary. In order to carry out the attack, it is necessary to hide the presence of the original AP. Cassola et al. accomplished this using a state-of-the art setup costing more than

$3600, which enabled them to selectively jam all beacons and probe responses, making the real AP invisible to nearby devices [34].

1.4 A Brief History of RC4

The RC4 algorithm is one of the most widely used stream ciphers. For example, in 2013 it was used in more than half of all TLS connections in an attempt to mitigate several attacks against CBC-mode encryption schemes [5, 6, 33, 46]. Additionally, RC4 is used in WEP and WPA-TKIP to encrypt packets [192]. Because of its popularity, it plays an important role in this dissertation. Therefore we will briefly go over the history of RC4, and highlight early (but ineffective) warnings that indicated this reliance on RC4 was far from ideal.

1.4.1 A secret computer code

Rivest designed RC4, short for Rivest Cipher 4 or Ron's Code 4, in 1987 when working for the company RSA Data Security. It was added to RSA's cryptographic library, and its first commercial usage was in Lotus Notes [143]. While initially a trade secret, it was anonymously posted to the Cypherpunks mailing list on 9 September 1994 [8]. Five days layer, the leaked source code was also posted on the `sci.crypt` Usenet newsgroup by David Sterndark [162]. Though it's now common practice to share the description of RC4 —most works that analyze RC4 contains its complete description— this was considered a bold move at the time. Indeed, its anonymous posting through the `jpunix` remailer caused a brief panic for John Perry, whom at the time was responsible of running the remailer. Someone informed his employer that John's remailer was being used to "send copy-written software as well as encrypted software out of the country" [130]. In response, John shutdown the remailer for a few days. Additionally, in the post to Usenet, David wrote:

> `I am shocked, shocked, I tell you, shocked, to discover that the cypherpunks have illegaly and criminally revealed a crucial RSA trade secret and harmed the security of America by reverse engineering the RC4 algorithm and publishing it to the world.`

That people living in the US were genuinely concerned with posting the source code of RC4 can also be seen in the reluctance to post it, under their real name, to the `sci.crypt` group. For instance, Bruce Schneier was not willing to repost the code at the time [152], and another member was only comfortable with

posting the headers of the original leak so people could find the leaked code themselves [108].

The leaking of RC4 even got picked up by The New York Times [102] and The Wall Street Journal [81], where they referred to it as an act of business espionage. The response of the RSA Data Security executives to the leak was [102]:

> RSA considers this misappropriation to be most serious. Not only is this act a violation of the law, but its publication is a gross abuse of the Internet.

In reality the leaking of RC4 is a good example of Kerckhoffs's principle: in a properly designed cryptographic system, security should not depend on the secrecy of an algorithm, but only on the secrecy of the key.

The name RC4 was trademarked by RSA Data Security in 1993 [148]. To avoid trademark issues, modern standards often use ARCFOUR or ARC4 (standing for alleged RC4) to refer to RC4 [79].

1.4.2 Known flaws and early warnings

In January 2006, RFC 4345 was published, which added a new cipher mode to SSH where RC4 is used but the first 1536 bytes of keystream must be discarded [70]. These discarded bytes must be kept secret. Furthermore, they recommend that RC4 should not be used when the same secret is repeatedly encrypted [70, §5]:

> Weak distinguishers can operate on any part of the keystream, and the best ones, described in (Fluhrer and McGrew, [56]) and (Mantin, [56]), can use data from multiple, different keystreams. A consequence of this is that encrypting the same data (for instance, a password) sufficiently many times in separate Arcfour keystreams can be sufficient to leak information about it to an adversary. It is thus recommended that Arcfour [..] not be used for high-volume password-authenticated connections.

Interestingly, this warning appears to have correctly anticipated the recent type of attacks on RC4 [5, 60, 182], though the RFC focusses on attacks against SSH instead of HTTPS. It is also worth noting that RC4 was explicitly mentioned as *not* being FIPS-approved for use in TLS implementations [35].

More intriguingly, in a StackExchange question on August 2012, a user by the pseudonym of `lxgr` also accurately predicted the recent attacks on RC4. He

was inspired by the RFC mentioned above, and asked whether it is possible to decrypt a password that is repeatedly transmitted over an IMAP or SMTP connection [93]. More generally, he was thinking about "protocols that always send the password at a well known offset". Roughly a week later Scott Fluhrer answered this question under his `poncho` pseudonym[1], and estimated that 8 billion encryptions of a password are sufficient to recover it. Unfortunately, he did not specify how he derived this estimate, only that he "was able to successfully recover the exact password by analyzing the statistics for the streams". One year later AlFardan et al. [5] showed that roughly 2^{33} encryptions are sufficient to recover a 16-byte secret. This appears to be in line with the prediction of Fluhrer. And in 2015, the attack scenario against IMAP passwords was worked out by Garman et al., showing around 2^{26} encryptions of a password are sufficient to recover it with high probability [60].

The above examples demonstrate that, at the time, a careful evaluation of research results on RC4 showed the cipher should be avoided in practice. However, most chose to ignore these warnings, disregarding the weaknesses as purely theoretic and academic. In practice, they claimed, an attack would be infeasible (some selected examples of such claims are [132, 133, 141]).

1.5 Other Contributions

The results presented in this dissertation are a subset of the research that has been performed over the last four years. In particular, this dissertation contains my works that are related to the security of WPA-TKIP and TLS. Other works and ongoing projects are listed below, along with a brief description of the obtained results.

Stateful Declassification Policies for Event-Driven Programs

This work proposes a novel mechanism for enforcing information flow policies with support for declassification on event-driven programs. The declassification policy specifies for each confidential event what information in the event can be declassified directly. Additionally, it can specify that aggregate information computed over all confidential events seen so far can be declassified. It is shown that such declassification policies are useful in the context of JavaScript web applications. An enforcement mechanism is presented and implemented in a browser, and its soundness and precision is proven. This work was conducted

[1] Scott Fluhrer also used the `poncho` pseudonym in the Usenet sci.crypt newsgroup.

in collaboration with Willem De Groef, Dominique Devriese, and Tamara Rezk from INRIA, France, under the supervision of Frank Piessens.

Publication data:

> M. Vanhoef, W. De Groef, D. Devriese, F. Piessens, and T. Rezk. "Stateful declassification policies for event-driven programs". In: *Proceedings of the 27th IEEE Computer Security Foundations Symposium (CSF '14)*. IEEE, July 2014, pp. 293–307

Why MAC Address Randomization is not Enough: An Analysis of Wi-Fi Network Discovery Mechanisms

This work presents several novel techniques to track (unassociated) mobile devices by abusing features of the Wi-Fi standard. In other words, it shows that using random MAC addresses, on its own, does not guarantee privacy. This work was conducted under the supervision of Frank Piessens, and in collaboration with the following researchers from INRIA, France: Célestin Matte, Mathieu Cunche, and Leonardo Cardoso.

The weaknesses that we discovered in Windows 10's implementation of MAC address randomization have been reported to Microsoft, and they are in the process of addressing these issues. Results of this work have also been presented by Mathieu Cunche at the IEEE working group responsible for developing recommended practices and privacy considerations for 802 technologies [178]. Our findings are currently being passed on to the 802.11 working group, meaning the issues we discovered will likely be addressed in an upcoming 802.11 standard.

Publication data:

> M. Vanhoef, C. Matte, M. Cunche, L. Cardoso, and F. Piessens. "Why MAC Address Randomization is not Enough: An Analysis of Wi-Fi Network Discovery Mechanisms". In: *Proceedings of the 11th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS '16)*. ACM, May 2016, pp. 413–424

Defeating MAC Address Randomization Through Timing Attacks

This work presents an attack to defeat Wi-Fi-based MAC address randomization purely based on the timing of network scans. In particular, it relies on the inter-frame arrival time of probe requests, which forms a rough signature of

a device. The main research effort of this work was done by Célestin Matte, who collaborated with me, Franck Rousseau (from Grenoble INP, France) and Mathieu Cunche.

Publication data:

> C. Matte, M. Cunche, R. Franck, and M. Vanhoef. "Defeating MAC Address Randomization Through Timing Attacks". In: *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '16)*. July 2016

Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys

This work analyzes the generation and management of 802.11 group keys. These keys are used to protect broadcast and multicast frames. It is argued that the random number generator proposed in the 802.11 standard may not provide a sufficient amount of entropy, and this argument is confirmed by predicting the generated groups keys on several platforms. Additionally, we found that an adversary can force usage of RC4 to encrypt the group key when it is being transferred to clients in the 4-way handshake. Finally, we demonstrate that knowledge of the group key is sufficient to decrypt and inject both broadcast and unicast traffic. This research was conducted under the supervision of Frank Piessens.

Publication data:

> M. Vanhoef and F. Piessens. "Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys". In: *Proceedings of the 25th USENIX Security Symposium (USENIX Security '16)*. USENIX Association, Aug. 2016

Request and Conquer: Exposing Cross-Origin Resource Size

This work explores various techniques that can be employed to reveal the size of resources hosted on the web. Several design flaws in the storage mechanisms of browsers are presented, which allow an adversary to determine the exact size of any cross-origin resource. Furthermore, a novel (link-layer) size-exposing technique against Wi-Fi networks is presented. The severity of these attacks are evaluated in multiple real-world attack scenarios. For instance, our attacks can be used to deanomize users, infer medical information on the victim, etc. The main research effort of this work was done by Tom Van Goethem, with whom I collaborated under the supervision of Frank Piessens and Wouter Joosen.

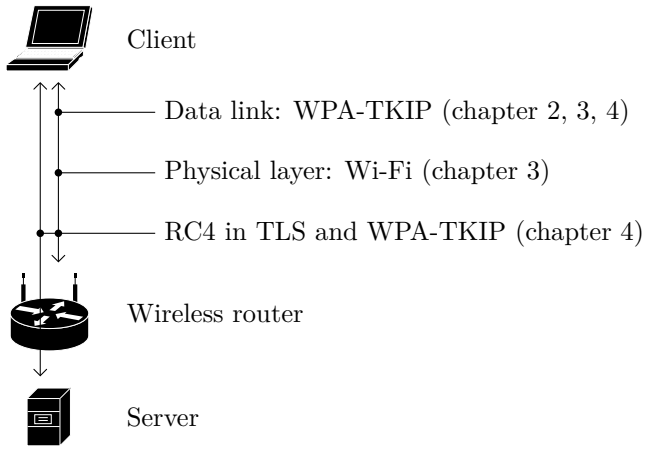

Figure 1.2: Visualization of the outline of this dissertation.

Publication data:

T. Van Goethem, M. Vanhoef, F. Piessens, and W. Joosen. "Request and Conquer: Exposing Cross-Origin Resource Size". In: *Proceedings of the 25th USENIX Security Symposium (USENIX Security '16)*. USENIX Association, Aug. 2016

1.6 Outline of the Dissertation

The remainder of this dissertation is structured as follows (see Fig. 1.2).

Chapter 2: Practical Verification of WPA-TKIP Vulnerabilities

In this chapter we study WPA-TKIP and present three attacks against it. The first attack is a Denial of Service attack that requires the injection of only two frames every minute. The other two attacks allow an attacker to decrypt and inject a select number of packets. We demonstrate the practicality of these attacks by performing a portscan on any client, and by decrypting arbitrary packets sent towards a client. In the next two chapters, we rely on these injection and decryption attacks to highlight the severity of weaknesses in WPA-TKIP.

Chapter 3: Advanced Wi-Fi Attacks Using Commodity Hardware

In this chapter our goal is to attack WPA-TKIP when used as a group cipher, i.e., when used to encrypt broadcast or multicast packets. In order to accomplish this, we require a man-in-the-middle position that allows us to reliably manipulate encrypted Wi-Fi traffic. To obtain such a position, we first study the security guarantees of the Wi-Fi protocol at the physical layer, and show that we can break certain assumptions by modifying the firmware of commodity Wi-Fi cards. This enables use to implement a reliable channel-based man-in-the-middle attack using only commodity Wi-Fi cards. We then demonstrate how this position can be used to break WPA-TKIP when it is used to protect broadcast or multicast traffic.

Chapter 4: Breaking RC4 in WPA-TKIP and TLS

In this chapter we search for new biases in the keystream of RC4 using statistical hypothesis tests. Then we develop novel techniques that exploit these biases to recover a repeatedly encrypted secret. We show how these techniques can be applied to attack WPA-TKIP and TLS. In the attack against WPA-TKIP we first generate a large number of identical packets. This packet is then decrypted, allowing us to recover the MIC key, which can be used to carry out the injection and decryption attacks presented in Chapter 2. We also attack RC4 when used in TLS, and show how to decrypt a secure cookie by generating, and subsequently capturing, roughly one billion encrypted HTTP requests containing the cookie. Finally, the performance of both attacks are evaluated in realistic scenarios.

Chapter 5: Conclusion

We end by summarizing the contributions of this dissertation, reflect on the obtained results, and mention open problems and interesting directions for future work.

Chapter 2

Practical Verification of WPA-TKIP Vulnerabilities

> "The right thing... what is it? I wonder, if you do the right thing... does it really make everyone happy?"
>
> — *Moon Children, The Legend of Zelda: Majora's Mask*

Abstract

We describe three attacks on the Wi-Fi Protected Access Temporal Key Integrity Protocol (WPA-TKIP). The first attack is a Denial of Service attack that can be executed by injecting only two frames every minute. The second attack demonstrates how fragmentation of 802.11 frames can be used to inject an arbitrary number of packets, and we show that this can be used to perform a portscan on any client. The third attack enables an attacker to reset the internal state of the Michael algorithm. We show that this can be used to efficiently decrypt arbitrary packets sent towards a client. We also report on implementation vulnerabilities discovered in some wireless devices. Finally, we demonstrate that our attacks can be executed in realistic environments.

Table 2.1: Number of Wi-Fi networks supporting a given encryption scheme, for several regions, in November 2013, December 2014, and April 2016. Note that one network can support multiple schemes.

	Region	Open	WEP	TKIP	CCMP	#Networks
2013:	Leuven	381	618	3 307	3 143	5023
	Heverlee	121	288	1 212	1 149	1 886
2014:	Leuven	237	289	3 850	5 096	5 993
	New Orleans	340	85	848	1 443	1 948
2016:	Leuven	231	176	4 364	6 963	7 586

Preface

This chapter was previously published as:

> M. Vanhoef and F. Piessens. "Practical verification of WPA-TKIP vulnerabilities". In: *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS '13)*. ACM, May 2013, pp. 427–436

The main research effort of this work was lead by Mathy Vanhoef with the guidance of Frank Piessens.

Three years after the publication of this chapter, the Wi-Fi Alliance changed its stance on deprecating, and eventually disallowing, WPA-TKIP. That is, effective 16 March 2016, they prohibit a certified device from offering a TKIP-only option in the device's *primary* interface [52]. However, the device can still be configured, through a secondary interface, to allow only TKIP. In practice this means only command-line tools can configure a router to only enable TKIP [37]. More worrisome, the Wi-Fi Alliance still allows (but discourages) the presence of a mixed network option in the main interface. In a mixed network, both TKIP and AES-CCMP can be used. Hence old devices can continue to use TKIP, while newer devices can use the more secure AES-CCMP. These changes in policy are surprising, since it effectively means TKIP is still allowed and supported.

Over the last three years we have also kept track of how many networks support WPA-TKIP. In particular, we surveyed wireless networks in both late 2014 and early 2016. The results of these surveys are shown in Table 2.1. In December 2014 we gathered data in two municipalities: first in Leuven (Belgium), and then in New Orleans (United States). All combined we detected 7941 networks, of which 7352 were using some form of encryption. Compared to our initial statistics from 2013, the number of encrypted networks that *only* allow TKIP

fell from 19% to 6%. However, the number of encrypted networks that support both TKIP and AES-CCMP merely decreased from 71% to 64%. In a second capture around April 2016 in Leuven, we detected 7 586 networks, and 7 355 of them were using encryption. We found that 59% of encrypted networks still support TKIP, though the number of encrypted networks that only allow TKIP decreased to 3%. Based on these observations we can conclude that, even though the Wi-Fi Alliance is trying to discourage the use of TKIP, more than half of all encrypted networks still support it.

2.1 Introduction

Modern wireless networks are based on the IEEE 802.11 set of standards. These networks have gained popularity over the years and are nowadays widely used in different scenarios, ranging from personal use to high-profile commercial use. Because of the nature of wireless transmission, special care must be taken to preserve the privacy and security of data sent over wireless networks. The original IEEE 802.11 standard supported a basic security algorithm called Wired Equivalent Privacy (WEP). Unfortunately WEP suffers from major design flaws and is considered completely broken [27, 57, 163].

An improvement of WEP is the Temporal Key Integrity Protocol (TKIP). Created as an intermediary protocol to the more secure CCMP, it was designed to run on existing WEP hardware [192, §11.4.1]. This affected many of the design decisions [51, 192]. Most notably it still uses WEP encapsulation and relies on a weak Message Integrity Check (MIC) algorithm called Michael [51]. Because the Michael algorithm provides inadequate security [72, 164, 193] countermeasures were added. TKIP and its countermeasures are explained in detail in Sect. 2.2.

Surprisingly, TKIP is still supported by a large number of networks. In Section 2.6.1 we report on an experiment, carried out in late November 2012, where we collected information about wireless network usage in two Belgian municipalities. We found that 71% of encrypted networks support TKIP. Furthermore, 19% of networks using encryption only allowed TKIP.

In this chapter we present a novel Denial of Service (DoS) attack on TKIP. Moreover, we take two ideas suggested in a paper by Beck [18] and significantly improve on them. In contrast with the paper of Beck, our improvements are also implemented and tested in practice. The first idea applies the known fragmentation attack on WEP [27] to TKIP. This allows an attacker to send an arbitrary number of packets to a client. As a proof of concept we implemented a port scanner. The second idea is to construct a prefix that resets the internal

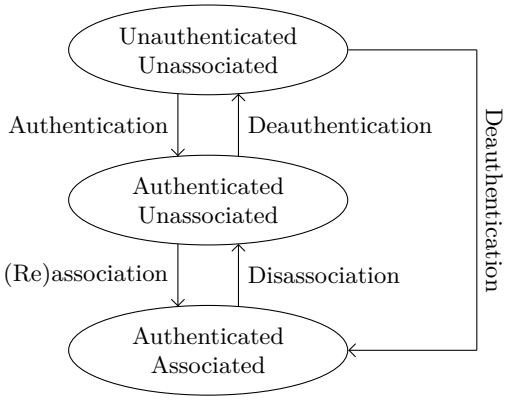

Figure 2.1: States a client can be in when connecting to a wireless network.

state of the Michael algorithm. We will show that this enables us to efficiently decrypt arbitrary packets sent towards a client. We also report on several vulnerabilities found in the implementation of some wireless adapters and drivers. All attacks are designed against Wi-Fi networks operating in infrastructure mode, and are tested when authentication is done using a passphrase and when using a personal username and password.

The remainder of this chapter is organised as follows. Section 2.2 describes the details of the TKIP protocol. In Section 2.3 we explain the Denial of Service (DoS) attack. Section 2.4 discusses our fragmentation and portscan attack. In Section 2.5 the Michael state reset and decryption attack is explained. Section 2.6 investigates whether TKIP is still supported in practice and discusses experimental evaluation of our attacks. Finally, we summarise related work in Sect. 2.7 and conclude in Sect. 2.8.

2.2 Temporal Key Integrity Protocol (TKIP)

This section describes the relevant parts of the IEEE 802.11 standard with a focus on the TKIP specification [192, §11.4.2]. We will also explain one of the first attacks on TKIP, called the Beck and Tews attack.

2.2.1 Connecting

A client connects to a wireless network by first authenticating and then associating with the Access Point (AP). A state diagram of this process is shown in Fig. 2.1. There are two authentication methods. The first one is called Shared Key authentication and was based on WEP. Unfortunately this method is inherently insecure. Nowadays only the second method is used, called Open System authentication. As the name implies it imposes no real authentication. It is just a formality, and if TKIP or CCMP is used the actual authentication will happen at a later stage. Once authenticated, the client sends an association request to the AP. This request includes the secure authentication and encryption protocol it wants to use. If the AP supports the requested protocols, the association is successful, and the AP informs the client that the association has completed.

Once authenticated and associated, a 4-way handshake is performed when using TKIP. The handshake negotiates the keys used by TKIP and is defined using IEEE 802.1X EAPOL-Key frames. This results in a pairwise transient key (PTK) that is shared between the AP and client. From the PTK a 256-bit temporal encryption key (TK) is derived, which is split into a 128-bit encryption key, and two 64-bit Message Integrity Check (MIC) keys: one for AP to client communication and one for client to AP communication. These keys are renewed after a user-defined interval, commonly called the rekeying timeout. Most APs by default use a timeout of 1 hour. After a key has been negotiated, the client and AP can send encrypted data frames to each other.

The client or AP can end the connection at any time by sending a disassociation or deauthentication message. The older versions of the 802.11 standard left these messages unprotected. This means an attacker can forge them and forcibly close the connection between an AP and client. Continuously injecting such forged deauthentication packets causes a DoS attack [123]. This attack is well known and is called the deauthentication attack. It can be prevented by enabling protected management frames, a feature introduced in the IEEE 802.11w amendment [192, §4.5.4.9].

2.2.2 Sender

When sending a TKIP frame first the MIC value of the MAC Service Data Unit (MSDU) is calculated. The purpose of the MIC is to protect both the integrity and authenticity of the message. Recall that the MSDU is the complete data packet that needs to be transmitted, which is fragmented into smaller MAC Protocol Data Units (MPDU) fragments if it is too big to be sent at once.

6	6	1	3	*var*
Destination	Source	Priority	Reserved	Payload

Figure 2.2: Input data given to the Michael algorithm. Destination and source represent MAC addresses. If the QoS extension is not used, priority is set to zero. Numbers denote the size of the field in bytes, where *var* defines a variable-length field.

	8	*var*	8	4
802.11 header	TSC	Data	MIC	ICV
⟵ unencrypted ⟶		⟵ encrypted ⟶		

Figure 2.3: Simplified format of an unfragmented TKIP frame.

The MIC is calculated over the MSDU by the Michael algorithm. Michael is a message authentication algorithm taking two inputs: the input data to calculate the MIC of, and a secret key. The input data given to the Michael algorithm is shown in Fig. 2.2. For calculating the MIC a different secret key is used for AP to client communication than is used for client to AP communication. Both MIC keys are derived from the PTK. The calculated MIC value is 8 bytes long. However, the Michael algorithm is not sufficiently secure. In particular it is possible to efficiently retrieve the MIC key given the data and calculated MIC value [164]. The designers realized this and included countermeasures in an attempt to mitigate potential attacks. These countermeasures are essential to our attacks and are discussed in detail in Sect. 2.2.3.

The MSDU concatenated with the MIC is fragmented into MAC Protocol Data Units (MPDUs) if necessary. At most 16 fragments (MPDUs) are supported. Each MPDU then undergoes WEP encapsulation. This is done so TKIP can be implemented on old WEP hardware. WEP encapsulation appends an Integrity Check Value (ICV) to the MPDU, which is simply a 32-bit CRC computed over the given data. Then it encrypts the packet using the RC4 stream cipher. The key used for encryption is called the WEP seed and is calculated by a mixing function that combines the temporal key (TK), transmitter MAC address, and the TKIP Sequence Counter (TSC). Figure 2.3 illustrates the final layout of an unfragmented TKIP frame.

Finally the resulting link-layer frame is constructed by adding the appropriate 802.11 headers. This includes the TSC, which is a replay counter that increases every time a MPDU frame is sent successfully. In Fig. 2.3 the TSC is given a size of 8 bytes, but in reality the TSC is only 6 bytes long. The remaining

bits of the field are used for other purposes or are reserved. To prevent replay attacks the receiver drops frames that are not received in order. That is, the counter must always be increasing though gaps are allowed.

2.2.3 Receiver

When receiving a TKIP frame the client or AP first checks if the TSC is in order. If not, the frame is silently dropped. It then proceeds by checking if the ICV is correct. If not, the frame is also silently dropped. Once all MPDUs are received they are reassembled into the original MSDU and its MIC value is verified. If it is correct the frame is accepted and the receiver updates its TSC replay counter, otherwise the TKIP countermeasures kick in. These countermeasures were added to detect active attacks against the weak Michael algorithm [192, §11.4.2.4.1]. One class of such active attacks involves injecting multiple forged packets in the hope at least one of them has a valid MIC. If such a packet is found the attacker learns the MIC key for the particular communication direction, as the MIC key can be derived when given the plaintext data and calculated MIC value.

The countermeasures are as follows [192, §11.4.2.4]:

- When a client receives a MSDU with an invalid MIC value it will send a MIC failure report to the AP.
- An AP receiving an invalid MIC value does not broadcast a MIC failure report but only logs the failure.
- If the AP detects two MIC failures within one minute all TKIP clients connected to the AP will be deauthenticated and the AP will not receive or transmit any TKIP-encrypted data frames for one minute. Once this minute is passed clients can reassociate with the AP and negotiate a new PTK.

2.2.4 Quality of Service extension

Quality of Service (QoS) enhancements for wireless traffic were first defined in the IEEE 802.11e amendment. Most modern APs support this amendment [116]. It defines 8 different channels, each having their own QoS needs. A channel is defined by its Traffic Identifier (TID) and is internally represented by 4 bits, making some devices actually support 16 different channels. For the implementation of our attacks we assume only 2 QoS channels exist, though

devices supporting more channels are also susceptible to our attacks. Tests showed that our assumption holds in practice. Additionally we note that by default most traffic is sent over the first QoS channel.

Essential for us is that each QoS channel has a separate TSC [192, §11.4.2.6]. This means that we can capture a packet transmitted on one QoS channel and replay it on another QoS channel having a lower TSC value. This enables an attacker to pass the TSC check, though he still has to pass the ICV and MIC checks in order to forge a message.

2.2.5 Beck and Tews attack

One of the first known attacks on TKIP was discovered by Beck and Tews [164]. It was a variation of the chopchop attack on WEP [66] and works by decrypting a packet one byte at the time. Because this attack is used as the basis for the fragmentation and Michael reset attack it will be explained in detail.

First we will explain the chopchop attack when applied to a TKIP packet. It begins by taking an encrypted packet and removing the last byte. Let C denote the obtained shortened encrypted packet. With high probability the ICV of C is invalid. However, it can be corrected if one knows the plaintext value of the removed byte. Correcting the ICV of the unencrypted shortened message can be represented by $M' = M \oplus D$, where M is the unencrypted shortened message, D is the correction being applied, $\oplus$ denotes the XOR operator, and M' is the unencrypted shortened message with a valid CRC. It has been proven that D only depends on the plaintext value of the removed byte [66]. Interestingly we can apply this modification directly to C. Letting K denote the keystream used to encrypt the packet we get that $C = M \oplus K$. We can now make the following derivation:

$$C' = M' \oplus K = (M \oplus D) \oplus K \tag{2.1}$$

$$= (M \oplus K) \oplus D \tag{2.2}$$

$$= C \oplus D\,. \tag{2.3}$$

The second equation follows from the associativity of the XOR operator. We see that C' is the encrypted shortened packet with a valid ICV, and that it can be obtained by directly applying the modification to the encrypted shortened packet C.

This technique can be used to decrypt TKIP packets sent towards the client as follows. An attacker tries all 2^8 possible values of the removed byte. For each guess the modification D is applied and the resulting packet is injected with

a different priority y. Assuming that QoS channel y has a lower TSC it will pass the TSC check, and the client will decrypt the packet. Then the ICV is verified. If the guess of the attacker was wrong the ICV is invalid and packet will silently be dropped (without generating a MIC failure). On the other hand, if the guess was correct, the ICV will be valid. However, with high probability the MIC value of the shortened packet will be wrong. As a result the client sends a MIC failure report. Thus a correct guess can be detected by listening for the corresponding MIC failure. Note that an AP isn't vulnerable to the attack because it never sends MIC failure reports. To avoid triggering the TKIP countermeasures at most one byte can be decrypted each minute.

Because we must wait one minute after decrypting a byte, it is infeasible to decrypt all bytes using this method. Instead the Beck and Tews attack targets an ARP reply packet and decrypts only the ICV and the MIC value. This takes on average 12 to 15 minutes. The remaining content of the packet is guessed. A particular guess can be verified by checking if the calculated ICV of the predicted packet equals the decrypted ICV. If they match, the guess is very likely correct. Once the ARP reply has been decrypted we can use the inverse Michael algorithm to calculate the MIC key used for AP to client communication [164]. Combined with the keystream of the decrypted ARP reply, an attacker can now forge 3 to 7 packets having a length smaller than or equal to the ARP reply. The precise number of packets that can be forged depends on the number of supported QoS channels.

2.3 Denial of Service

This section describes a novel attack we discovered. When closely inspecting the QoS extension to TKIP we notice that the keystream is independent of the priority (i.e., QoS channel) used to transmit the frame. On the other hand the calculated MIC value does depend on the priority of the MSDU. Say that we capture a packet sent with priority x and replay it with a different priority y. Assuming QoS channel y has a lower TSC it will pass the TSC check, and the receiver will use the correct keystream to decrypt the packet. As a result it will also pass the ICV check. However, the changed priority will cause the receiver to expect a different MIC value. Hence a MIC failure occurs. Replaying this packet a second time will trigger a second MIC failure. After these two MIC failures the AP will shut down all TKIP traffic for 1 minute. Repeating this process every minute will prevent any TKIP protected communication, effectively causing a DoS. If the network does not use QoS we can forge the QoS header when replaying the packet. As discovered by Morii and Todo, most clients will not check whether the network is actually using QoS and

simply accept the packet [116]. Therefore the only requirement is that one or more clients *support* the QoS extension, which is true for most modern wireless adapters [116].

To implement the attack we had to patch the compat-wireless drivers of Linux. The original driver modified the QoS header when in monitor mode, which interfered with our attack. Monitor mode is a feature supported by some wireless adapters and drivers allowing one to capture all wireless traffic. It also enables injection of arbitrary 802.11 frames. Our tool monitors the traffic of a network and shows whether it supports TKIP or QoS. Additionally it shows a list of connected clients and basic statistics such as the cipher suite it is using, number of MIC failures, number of association failures, etc. When a vulnerable TKIP packet is captured its priority is changed and the packet is replayed. If the network isn't using QoS our tool will forge the QoS header. We show in Sect. 2.6 that our implementation was found to be very reliable.

Our attack appears to be one of the more effective DoS attacks that can be launched against a network using TKIP. The simplicity of the attack not only makes it easy to implement and debug, it also assures it is applicable in many real world environments. In Section 2.7 a thorough comparison is given to currently known DoS attacks. Disabling the TKIP countermeasures prevents the attack. However, most APs do not provide this option, and with good reason. As mentioned in Sect. 2.2.3 the countermeasures were included to detect active attacks against the weak Michael algorithm [51].

Another option to prevent the attack is to make the keystream dependent on the priority. This entails a change in the protocol, meaning all devices implementing TKIP would have to be updated. Though preventing the attack, such a modification does not appear feasible in practice.

2.4 Injection of More and Bigger Packets

The Beck and Tews attack allows forging 3 to 7 packets of at most 28 bytes [164]. Our goal is twofold: we want to inject both more and bigger packets. In this section we assume the Beck and Tews attack has already been executed, meaning we have obtained the MIC key for AP to client communication. Our technique is similar to the fragmentation attack on WEP [27] and an improvement of a suggested attack by Beck [18]. To the best of our knowledge, this is the first time such an attack on TKIP has been implemented and proved to be possible.

2.4.1 Exploiting fragmentation

Since we know the MIC key, sending additional packets only requires obtaining new keystreams. Recall that each TSC corresponds to a different keystream. Because RC4 is used, XORing an encrypted packet with its corresponding unencrypted packet unveils the keystream used during encryption. Hence, finding new keystreams reduces to predicting the content of encrypted packets.

All 802.11 packets start with a LLC and SNAP header of 8 bytes. If we know whether it is an ARP, IP, or EAPOL packet, this header can be predicted. Fortunately we can identify ARP and EAPOL packets based on their length, and can consider everything else to be IP. The first byte of an IP packet consists of the version and the header length, which is almost always equal to 0x45. The second byte contains the Differentiated Services Field, and can be predicted based on the priority of the 802.11 frame [123]. Finally the next 2 bytes, representing the length of the IP packet, can be derived from the length of the 802.11 frame. As a result we can predict the first 4 bytes of an IP packet. Predicting the first 14 bytes of ARP packets is also possible. These bytes consist of the hardware type and size, protocol type and size, request type, and finally the MAC address of the sender. All these fields can be predicted in an IPv4 network. Because EAPOL packets are rarely transmitted we simply ignore them. Our predictions can then be XORed with the encrypted packets, revealing the first 12 bytes or more of the keystream.

These short keystreams are combined using fragmentation at the 802.11 layer. As mentioned in Sect. 2.2.2 the 802.11 protocol allows a frame to be fragmented into at most 16 MPDUs. Each MPDU must have a unique TSC value and is encrypted with the keystream corresponding to that TSC. All fragments must include an ICV, though the MIC value is calculated over the complete MSDU and can be spread over multiple MPDUs. Using fragmentation to combine the keystreams we can inject packets with 112 bytes of payload. Assuming there is enough traffic on the network, we can inject an arbitrary number of packets. Compared to the Beck and Tews attack this is a significant improvement.

2.4.2 Implementation: performing a portscan

Implementing the fragmentation attack required patching the compat-wireless drivers of Linux. The original driver failed to correctly inject MPDUs, which caused the attack to fail. Using our patched driver we successfully tested the fragmentation attack against several devices, and found that our heuristics to predict the first bytes of packets were very accurate (see Sect. 2.6). To reduce packet loss we also detect whether the client acknowledged receiving the MPDU,

otherwise the MPDU is retransmitted. As a proof of concept we implemented a port scanner. This requires injecting a large number of packets and is thus ideal to test our fragmentation attack.

The port scanner is given a file containing the ports to scan and works by injecting TCP SYN request to each port. The SYN packets do not contain any options, hence there is enough keystream to inject them. The encrypted TCP SYN-ACK reply is detected by its length. In practice most packets are larger than a SYN-ACK packet, meaning it has a distinctive short length. After scanning a port a TCP RST packet is always sent, even if no SYN-ACK was detected. This is done to prevent the client from retransmitting a potentially undetected SYN-ACK packet. Note that if the replies of the client can be sent to an IP under our control, we have a bidirectional communication channel, meaning we can connect to open ports.

The performance of the port scanner depends on several factors. First, recall that it requires knowledge of the MIC key. Hence an attacker must first execute the Beck and Tews attack, which on average takes between 12 to 15 minutes. Second, the attacker must have access to a sufficient amount of keystream in order to inject all TCP packets. This means there must be enough traffic on the network. Third, the performance depends on the quality of the wireless connection between the attacker and the victim. In particular, the attacker will have to frequently retransmit packets if this connection is bad, reducing the execution speed of the attack. Assuming the Beck and Tews attack has already been executed, that there is enough keystream, and that we have a reliable connection, scanning ports is fairly fast. That is, under these conditions we were able to scan 500 ports within roughly three minutes.

The attack can be mitigated by preventing the Beck and Tews attack. This means disabling the client from sending MIC failure reports, or using a short rekeying time of 2 minutes or less [164].

2.5 Decrypting Arbitrary Packets

In this section we describe a state reset attack on the Michael algorithm and we show how it can be used to decrypt arbitrary packets sent towards the client.

2.5.1 The Michael algorithm

The state of the Michael algorithm is defined by two 32-bit words (L, R) called left and right. To calculate the MIC value of an 802.11 packet the state is first

initialised to the MIC key. Then the data shown in Fig. 2.2 is processed, which is padded so that its length is a multiple of 4 bytes. Finally the calculation is finalised and the resulting MIC value is outputted. All data is processed in 32-bit words by a block function.

The block function $B(L, R)$ is an unkeyed 4-round Feistel-type construction, taking as input a Michael state and returning a new state [192, §11.4.2.3.3]. When processing an input word M the next state is given by $B(L \oplus M, R)$. We will let L' stand for $L \oplus M$. The block function can be inverted [193], and its inverse is denoted by B^{-1}. Note that we can predictably influence L at the start of the block function using M. For convenience the notation $B((L, R), M)$ is used to represent $B(L \oplus M, R)$, denoting the new internal state after processing M.

2.5.2 Michael state reset

If the Michael state ever returns to the initial state, all data processed so far has no influence on the MIC value. This idea can be used to construct a prefix packet which resets the state, allowing us to append a packet whose MIC value is calculated only over the appended data. As suggested by Beck [18] this could be done by appending two *magic words* to the prefix. These so-called magic words are ordinary 32-bits data words, but chosen in such a way so they reset the internal state of the Michael algorithm. Using them an attacker can append any encrypted packet to the prefix without invalidating the MIC value of the complete packet. Unfortunately Beck didn't provide a thorough theoretical analysis. We generalise the problem to finding a list of magic words that will transform a start state (L_s, R_s) to an end state (L_e, R_e).

We could try to use just one magic word M_1 to reset the state. In that case $B(L_s \oplus M_1, R_s)$ must return (L_e, R_e). Assuming that processing a random word M using the block function results in a random state, a guess for M_1 can be modelled as a Bernoulli trial with a success probability of 2^{-64}. Since we can try at most 2^{32} values for M, finding a solution reduces to having the first success after 2^{32} trials. This follows a geometric distribution. We get

$$\Pr[X \leq 2^{32}] = 1 - \left(1 - 2^{-64}\right)^{2^{32}} \approx 2.328 \cdot 10^{-10}\,, \tag{2.4}$$

where X follows a geometric distribution with a success probability of 2^{-64}. Such a low chance of finding a solution is unusable.

A better option is to use two magic words, denoted by M_1 and M_2. This gives us one intermediate state (L_i, R_i) to work with (see Fig. 2.4). We begin by calculating $B^{-1}(L_e, R_e) = (L'_i, R_i)$. Note that we cannot calculate L_i because

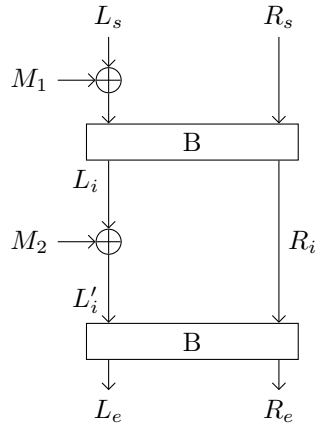

Figure 2.4: Visualization of all the variables used when resetting the state of the Michael algorithm using the two words M_1 and M_2.

M_2 is still unknown. Nevertheless, this teaches us the required value for R_i. Then we brute force the first magic word M_1. For each possible value we apply the block function. If we obtain the required value for R_i we have found a valid intermediate state (L_i, R_i), since using the guessed value for M_1 and setting M_2 to $L_i \oplus L'_i$ results in a solution:

$$B(B((L_s, R_s), M_1), M_2) = B(L_i \oplus M_2, R_i) \tag{2.5}$$

$$= B(L'_i, R_i) \tag{2.6}$$

$$= (L_e, R_e)\,. \tag{2.7}$$

The first equation is trivial. The second equation follows from our choice of M_2. Finally, the third equation follows from the calculation $B^{-1}(L_e, R_e) = (L'_i, R_i)$.

A solution is found if the guess for M_1 results in the required value for R_i, which has a probability of 2^{-32}. Similar to the previous case, the probability of finding a solution can be modelled by a geometric distribution:

$$\Pr[Y \leq 2^{32}] = 1 - \left(1 - 2^{-32}\right)^{2^{32}} = 0.6321\ldots\,, \tag{2.8}$$

where Y follows a geometric distribution with a success probability of 2^{-32}. This implies that roughly 27% of the time 2^{32} calculations are performed yet no solution is found. As an experiment we ran 50 000 runs where random Michael states had to be connecting using two magic words. In 31 518 runs a solution was found, resulting in a success probability of 63.036%. This closely matches

our analyses. On an 3.10 GHz Intel Core i5-2400 it took on average 11.14 seconds to find a solution, with a standard deviation of 6.33 seconds.

Another strategy is to perform a meet-in-the-middle attack using three magic words M_1, M_2, and M_3. This gives us two intermediate states (L_{i1}, R_{i1}) and (L_{i2}, R_{i2}) to work with. Again we start by calculating $B^{-1}(L_e, R_e) = (L'_{i2}, R_{i2})$. Next we take 2^{16} random values for the first magic word and compute the list of resulting intermediate states (L_{i1}, R_{i1}). We then try to brute-force the second magic word by applying the reverse Michael algorithm to the state (L'_{i2}, R_{i2}). If the resulting value for R_{i1} is in the list of earlier calculated states we have found a solution. Let L_{i1} be the value accompanying R_{i1}, then the magic words M_1, $M_2 = L_{i1} \oplus L'_{i1}$, and $M_3 = L_{i2} \oplus L'_{i2}$ provide a solution:

$$\begin{aligned} B(B(B((L_s, R_s), M_1), M_2), M_3) & \\ &= B(B(L_{i1} \oplus M_2, R_{i1}), M_3) && (2.9) \\ &= B(L_{i2} \oplus M_3, R_{i2}) && (2.10) \\ &= (L_e, R_e)\,. && (2.11) \end{aligned}$$

The first equation is trivial. The second and third follow from our choice of M_2 and M_3, and the usage of the reverse block function to calculate L'_{i1} and L'_{i2}.

The probability that the result of a guess for M_2 is in the list is 2^{-16}. The probability of finding a solution can again be modelled as a geometric distribution. We get

$$\Pr[Z \leq 2^{32}] = 1 - \left(1 - 2^{-16}\right)^{2^{32}} \approx 1 - 7.2 \cdot 10^{-28463}\,, \tag{2.12}$$

where Z follows a geometric distribution with a success probability of $p = 2^{-16}$. Practically this shows that a solution will always be found. The average number of required guesses for the second word is $\mathrm{E}[Z] = 1/p = 2^{16}$. As an experiment we ran 50 000 runs where random states had to be connecting using three magic words. In all runs a solution was found. It took on average $65\,814 = 2^{16.017\ldots}$ guesses for M_2 until a solution was found, with a standard deviation of 66 407 guesses. On our 3.10 GHz Intel Core i5-2400 this corresponded to an average running time of 2.96 milliseconds. Though requiring more magic words, these results are significantly better than the previous two cases.

2.5.3 Decryption attack

Our goal is to decrypt arbitrary packets sent towards the client. We will accomplish this by appending the targeted packet to a specially crafted prefix.

The prefix will simulate the behaviour of a ping request, making the client echo back the appended data. We construct the prefix such that the reply is sent to an IP under our control, meaning we will receive the plaintext content of the targeted packet, effectively decrypting it. To assure that the MIC value of the constructed packet is correct we will apply the Michael state reset attack to the ping-like prefix. This allows us to append the targeted packet without invalidating the MIC value. The resulting frame is sent to the client using the fragmentation attack.

Contrary to the suggestion by Beck [18] we cannot use an ICMP ping request as the prefix. This is because it includes a checksum calculated over the header and the data section. But since we do not know the plaintext data of the full packet, we cannot calculate a correct checksum. Instead we will construct a UDP prefix, where specifying a checksum is optional. Sending a UDP packet to a closed port results in an ICMP destination unreachable reply containing the first 8 bytes of the UDP packet. However, on Windows, Linux, and Android the ICMP unreachable reply contains a full copy of the original UDP packet. So when targeting these operating systems the client will reply with the complete content of our constructed packet. In particular this includes the plaintext content of the targeted packet. Even large packets can be quickly decrypted using this method.

Again we created a proof of concept tool in Linux. It listens for packets sent towards the client. Once a vulnerable packet has been captured, the UDP prefix is constructed and the Michael state reset attack is applied. The resulting UDP prefix is transmitted using the fragmentation attack, followed by the targeted packet (which is marked as the final fragment of the MSDU). In practice this means the final fragment will usually be bigger than all previous fragments. Though this is not allowed by the 802.11 specification, nearly all devices will accept the packet (see Sect. 2.6). The client will send an ICMP unreachable reply to the source IP address of the injected UDP prefix. This reply includes the prefix, magic bytes, and the unencrypted data of the targeted packet. Among other things, this allows an attacker to decrypt a TCP packet, learn the sequence numbers currently used in this TCP connection, and hijack the TCP stream to inject arbitrary data [71]. As a consequence, malicious data could be injected when the client opens a website. Again the attack can be mitigated by preventing the Beck and Tews attack.

2.6 Experiments

In this section we begin by investigating whether TKIP is still supported in practice. Then we evaluate to which extent devices adhere to the relevant aspects of the 802.11 standard, we report on implementation vulnerabilities discovered in some wireless devices, and we discuss how our findings impact the attacks. Finally our attacks are tested in realistic settings.

2.6.1 Networks supporting TKIP

Some new routers (for instance the Belkin N300 router) do not support TKIP any more, in accordance with the security roadmap of the Wi-Fi Alliance. The Wi-Fi Alliance tests products and hands out certifications if they conform to certain standards. Their roadmap specified that, as of 2011, new APs are no longer allowed to support a TKIP only option [55]. Even mixed mode, which simultaneously allows TKIP and CCMP in the same network, is no longer a requirement. Finally, in 2014 TKIP was supposed to be disallowed completely. Based on this one would think that TKIP is no longer widely supported. Surprisingly, we found the opposite to be true.

To investigate whether TKIP is still supported in practice we surveyed wireless networks in two Belgian municipalities (Leuven and Heverlee). Detecting networks was done using passive scanning, consisting of monitoring wireless traffic for beacon frames. These frames contain all information necessary to connect to a wireless network. In particular it includes the name of the network, the MAC address of the AP, and the encryption schemes supported by the network. During an initial test we found that active scanning detected few additional networks (less than 6%), so in an attempt to prolong battery life only passive scanning was used.

Several trips, of around an hour long, were made on foot while scanning for networks. The raw capture was written to file and later analyzed for beacon frames using a custom tool. This approach allowed us to make improvements to our tool after collecting the raw captures. We uniquely identified networks by their name. This is necessary because one network can be advertised by multiple APs. However we also encountered several APs advertising a network named after a vendor or product. These are default network names used by a particular device and do not represent the same network. Therefore we treated each of these APs as a unique network. Similarly there were several APs advertising a network with an empty name. These APs were also treated as an unique network.

Table 2.2: Number of Wi-Fi networks supporting a given encryption scheme for several regions. Note that one network can support multiple schemes.

Region	Open	WEP	TKIP	CCMP	#Networks
Leuven	381	618	3 307	3 143	5 023
Heverlee	121	288	1 212	1 149	1 886

In total we detected 6 803 unique networks. The number of networks supporting a particular encryption scheme is shown in Table 2.2. Note that a handful of networks were present both in Leuven and Heverlee. We found that 93% of the networks used encryption, and that 66% supported TKIP. When considering only encrypted networks, 71% of them supported TKIP. Additionally, 19% of networks using encryption only allow TKIP. We believe the reason so many networks still support TKIP is because most routers, when configured to use WPA2, by default use mixed mode (allowing both TKIP and CCMP). We even observed that WEP is still used by 14% of encrypted networks.

2.6.2 Adherence to the 802.11 specification

The wireless adapters in Table 2.3 were tested for implementation details impacting our attacks. While doing this we encountered several vulnerabilities present in some wireless devices, and decided to test for their presence on all devices. Wireless routers and mobile devices were tested using their default configurations. Open source wireless router firmware was tested on the Asus RT-N10. Laptops and USB wireless adapters were tested on Linux using the compat-wireless 3.6.2-1-snp drivers, and on Windows using the default installed drivers.

For the DoS attack to work a client must send a MIC failure report when an invalid MIC has been detected, and the AP must shut down the network after two MIC failures. We found that all wireless adapters, in all configurations, implement this properly. As a result our DoS attack is applicable to all tested devices (see Table 2.3 column DoS). Additionally we tested the DoS on a network not using QoS, making our tool forge the QoS header. Again all adapters were vulnerable to the attack, confirming results by Todo et al. [168].

Fragmentation support has been tested for several properties. We found that the Belkin F5D7053 adapter on Windows incorrectly implemented fragmentation. Sending a fragment to this adapter always resulted in a MIC failure, though it worked fine under Linux. All other devices supported fragmentation.

Table 2.3: Results of various tests on 14 wireless adapters. For laptop and USB adapters, L and W denote it only works on Linux or Windows, respectively. Yes means it works on both, no means it works on neither. Open Source router firmware was tested on the Asus RT-N10.

		DoS	Fragmentation				Replay	Pt.
			diff. size	eff. frag.	skip TSC	any MIC		
Laptop/USB:	Intel 4965AG	yes	yes	no	L	no	no	no
	Belkin F7D1102AZ	yes	yes	yes	yes	W	yes	L
	Belkin F5D8053	yes	L	L	L	L	yes	yes
	Alfa AWUS036h	yes	yes	yes	yes	W	W	W
	Ralink WA-U150BB	yes	yes	yes	yes	L	yes	W
Mobile Devices:	iPod MC086LL	yes	yes	no	yes	no	no	no
	iPad MC980NF	yes	yes	no	yes	no	no	no
Access Points:	Linksys WAG320N	yes	yes	yes	yes	no	no	no
	WRT54G 4.21.5	yes	yes	yes	yes	no	no	no
	Scarlet VDSL Box	yes	yes	no	no	yes	no	yes
	Cisco Aironet 1130AG	yes	yes	yes	yes	no	no	no
	Asus RT-N10 1.0.2.4	yes	yes	yes	yes	no	no	no
	Tomato 1.28	yes	yes	yes	yes	no	yes	no
	DD-WRT v24-sp2	yes	yes	yes	yes	no	no	no

An important property of fragmentation is whether the last fragment is allowed to be bigger than the previous fragments. Strictly speaking this is not allowed in the 802.11 standard [192, §9.5] yet we rely on this during the decryption attack. We found that all devices supporting fragmentation permitted this behaviour (see Table 2.3 column diff. size).

As mentioned in Sect. 2.2.3 the receiver should update its replay counter after reassembling the MSDU. This leaves open the possibility to send each fragment of an MSDU using the same TSC, keystream, and priority. If allowed, an attacker is then able to inject 16 fragments using only one keystream. In Table 2.3 column eff. frag. we see that this technique is indeed possible on several devices.

Finally we tested if fragments can be sent without using a sequential replay counter, i.e., whether we can skip TSC values. The 802.11 standard permits this behaviour for TKIP [192, §11.4.2.6], though some devices only accepted sequential TSC (see Table 2.3 column skip TSC).

Surprisingly we also discovered critical implementation flaws in some devices, particularly in wireless USB adapters. Several do not drop packets with an already used TSC, allowing an attacker to replay packets (see Table 2.3 column Replay). Another flaw is that certain devices do not verify the MIC value of a fragmented TKIP packet (see Table 2.3 column any MIC). This removes the requirement of first having to execute the Beck and Tews attack to obtain the MIC key. After all, if the MIC value is not verified we do not have to calculate it, meaning there is no reason to know the MIC key. On Windows the Ralink WA-U150BB drops fragmented packets with an incorrect MIC, but does not transmit a MIC failure. As mentioned in Sect. 2.3, dropping the packet but not sending a MIC failure can open the door for novel attacks on the MIC key.

More worrisome, we also encountered several wireless adapters that accepted unencrypted (plaintext) packets while connected to an encrypted network (see Table 2.3 column Pt.). In particular the vulnerability is present in the Windows drivers for the AWUS036h, a device popular for its wide reception and high transmission power. Another device susceptible to the attack was the Scarlet VDSL Box. This is a router that is handed out by the Belgium ISP Scarlet when a customer buys a VDSL internet connection. The reason this case is interesting is because the vulnerability has a higher impact when present on APs. An attacker could abuse the flaw to easily inject arbitrary traffic into the network, but also to send packets to the internet using the public IP of the victim. It might be an interesting future research direction to test for additional implementation flaws on a wider array of devices.

Table 2.4: Impact of our DoS attack on several Access Points. The term *Both* denotes that TKIP and CCMP traffic was disabled. Open source router firmware was tested on the Asus RT-N10.

Access Point	Protocols Disabled	
	WPA1	WPA2
Linksys WAG320N	Both	Both
WRT54G 4.21.5	TKIP	Both
Scarlet VDSL Box	Both	TKIP
Cisco Aironet 1130 AG	Both	Both
Asus RT-N10 1.0.2.4	Both	TKIP
Tomato 1.28	Both	Both
DD-WRT v24-sp2	Both	Both

2.6.3 Verifying our attacks

According to the 802.11 specification, the TKIP countermeasures must disable only TKIP traffic for one minute [192, §11.4.2.4]. However, most of the APs we tested disabled all wireless traffic, including CCMP protected traffic in mixed mode. When targeting these devices one single TKIP client allows an attacker to take down the complete wireless network. More precisely, the behaviour of an AP depends on whether WPA1 or WPA2 is being used (see Table 2.4). In practice, the difference between WPA1 and WPA2 is mainly between supported encryption schemes: WPA1 mandates TKIP support and optionally allows CCMP, while the reverse is true for WPA2. For example, the WRT54G only supports TKIP when using WPA1, while the Belkin N300 only supports WPA2 with CCMP.

For the fragmentation, portscan, and decryption attack to work we need to be able to accurately predict the first 12 bytes of encrypted packets. To test our prediction algorithm we monitored TKIP traffic generated by visiting several websites for 20 minutes. In total 36 643 data packets were captured, of whom 36 642 were predicted correctly. The single miss was an EAPOL packet, which we purposely ignored. Further, we weren't able to capture all packets as an attacker, meaning the keystreams corresponding to certain TSCs remained unknown. In total 5 680 packets were not captured, consisting of 13% of all traffic.

The fragmentation and portscan attack has been tested against Windows, Linux, iOS, and Android. When connected to a networking using TKIP, all were found to be vulnerable. Additionally we tested the attacks under two authentication mechanisms: the first being a shared passphrase and the second being a personal

login and password verified using PEAP-MSCHAP v2. In both situations the attacks were successful. These tests clearly demonstrate the reliability of the attacks.

For the decryption attack we found that, when connected to a network using TKIP, Windows, Linux, and Android were vulnerable. Mac OS X and iOS only include the first 8 bytes of the UDP packet in the ICMP unreachable reply, meaning the targeted packet is never included. Nevertheless, we did receive ICMP unreachable replies from all operating systems, meaning the Michael reset attack worked in all cases. Installing a firewall which blocks the ICMP unreachable replies prevents the decryption attack, though the Michael reset attack itself will remain possible.

2.7 Related Work

Several DoS attacks exist on wireless networks. Arguably the most well-known attack consists of forging deauthentication frames to either the client or AP [19]. This is possible because the deauthentication message is not protected using any keying material. Continuously sending them will result in a DoS attack. An advantage of our attack is that it requires replaying only two frames each minute to disconnect *every* client that is using TKIP. Hence our attack is stealthier and requires less power to execute. Furthermore, nowadays the deauthentication attack can be prevented by enabling protected management frames [192, §4.5.4.9]. Compared to the DoS attack of Glass and Muthukkumarasamy [62] our attack is easier to execute, since their attack requires a man-in-the-middle position. Another advantage is that our attack can be used to easily verify whether the client sends MIC failure reports, and thus to see if it is possible to perform the Beck and Tews attack [116]. The Beck and Tews attack could also be changed to a DoS attack. Once the first MIC failure has been detected, the corresponding packet could be injected a second time. However, on average 128 packets have to be injected before a MIC failure is triggered. This makes the attack easier to detect. It also isn't as easy to implement compared to our DoS attack, making it more difficult to execute in practice.

Könings et al. found DoS vulnerabilities at the physical and MAC layer of 802.11, some halting traffic for one minute with minimal packet injection [84]. Contrary to our DoS, their attacks do not simultaneously disconnect all clients connected to a network. Bicakci and Tavli give a survey on DoS attacks at the physical and MAC layer. The attacks they discuss require injecting a large number of frames [26].

One of the first attacks on TKIP was found by Beck and Tews and decrypts an ARP reply packet. As a result the MIC key for AP to client communication can be obtained [164] and a few small packets can be injected. It only works if the QoS extension is enabled and takes 12–15 minutes to execute. Halvorson et al. improved the attack, allowing larger packets to be injected [69]. However their technique does not allow injection of more packets, and takes longer to execute compared to our fragmentation attack. Morii and Todo came up with a modification removing the requirement that the AP must have enabled QoS [116]. They found that even if QoS was not used, clients will still accept and process a QoS frame. This allows an attacker to forge the QoS header if not present. In another paper Todo et al. managed to reduce the execution time of the Beck and Tews attack to 8–9 minutes [168].

The Michael algorithm was designed by Ferguson [51]. It was quickly revealed to be invertible by Wool [193]. Based on this Wool suggested a related message attack on TKIP. Huang et al. showed that Michael is not collision free and suggested a packet forgery attack [72].

Beck suggested the use of two magic words to reset the internal state of the Michael algorithm [18]. Unfortunately, no thorough theoretical analysis was provided, and no implementation in a practical setting was given. Based on the fragmentation attack on WEP by Bittau et al. [27], Beck also suggested using the fragmentation attack on TKIP, though no implementation was made to verify his ideas.

Wireless drivers have been tested before for vulnerabilities [31]. Most of the techniques focus on fuzzing with the aim of finding code injection attacks. However, only a limited amount of research has been done to find logical implementation flaws. In particular we found no previous work testing for replay attacks, seeing if it is possible to inject plaintext packets into an encrypted network, or testing whether the MIC of fragmented TKIP packets is verified.

Moen et al. have analyzed the key scheduling algorithm in TKIP [113]. They found that, given less than 10 RC4 packet keys, it is possible to recover the temporal key (TK) with a time complexity of $\mathcal{O}(2^{105})$ compared to a brute force attack with complexity $\mathcal{O}(2^{128})$. Sepehrdad et al. showed how to recover the TK using 2^{38} packets with a time complexity of $\mathcal{O}(2^{96})$ [156].

2.8 Chapter Conclusion

TKIP was designed as an intermediary solution to mitigate the existing flaws of WEP. We found that TKIP is still supported by a large number of networks.

Further, we showed that TKIP fails to provide sufficient security, by describing and implementing several new attacks. In addition, during our experiments, we identified several critical implementation vulnerabilities in several wireless devices. We conjecture that a more substantial test of these implementations will reveal more vulnerabilities.

Specifying a short rekeying interval prevents the fragmentation and decryption attack, unfortunately this does not prevent our DoS attack. To secure a wireless network it is strongly advised to only support the more secure CCMP.

Chapter 3

Advanced Wi-Fi Attacks Using Commodity Hardware

> "Don't touch the poster at the Game Corner!
> There's no secret switch behind it!"
>
> — *Team Rocket, Pokémon Yellow*

Abstract

We show that low-layer attacks against Wi-Fi can be implemented using user-modifiable firmware. Hence cheap off-the-shelf Wi-Fi dongles can be used to carry out advanced attacks. We demonstrate this by implementing five low-layer attacks using open source Atheros firmware. The first attack consists of unfair channel usage, giving the user a higher throughput while reducing that of others. The second attack defeats countermeasures designed to prevent unfair channel usage. The third attack performs continuous jamming, making the channel unusable for other devices. For the fourth attack we implemented a selective jammer, allowing one to jam specific frames already in the air. The fifth is a novel channel-based Man-in-the-Middle (MitM) attack, enabling reliable manipulation of encrypted traffic.

These low-layer attacks facilitate novel attacks against higher-layer protocols. To demonstrate this we show how our MitM attack facilitates attacks against

the Temporal Key Integrity Protocol (TKIP) when used as a group cipher. Since a substantial number of networks still use TKIP as their group cipher, this shows that weaknesses in TKIP have a larger impact than previously thought.

Preface

This chapter was previously published as:

> M. Vanhoef and F. Piessens. "Advanced Wi-Fi attacks using commodity hardware". In: *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC '14)*. ACM, Dec. 2014, pp. 256–265

The main research effort of this work was lead by Mathy Vanhoef with the guidance of Frank Piessens.

The source code of the tools that were created for this publication were publicly released [175, 176]. Over the last two years this code has also been kept up-to-date, assuring it still works on modern Linux distributions. Currently our release supports Linux kernels 3.0 up to and including kernel 4.4. As mentioned in the chapter, the public release does not contain the code for the constant jammer, as this could easily be used for malicious purposes to disrupt networks. People wanting to experiment with the constant jammer have to request the code privately. Overall we received more than 25 requests for the constant jammer implementation. Most requests where by researchers, e.g., from MIT, INRIA, Unversity of Luxembourg, Griffith University, etc., who were studying the impact of jammers on a wireless network, and how to defend against jamming attacks. There were also a significant number of requests from industry (some examples being Cisco, PwC, BT, Netscout, and so on) who were interested in using the tool for various purposes.

To disseminate our findings this work has also been presented at BruCON 2015, where the focus was on the low-layer Wi-Fi attacks. After this presentation the work was picked up by several online news magazines and blogs. For example, it was mentioned by Bruce Schneier on his On Security blog [150], by The Register [128], SC Magazine [107], etc.

As mentioned in Chapter 2, more than half of all encrypted Wi-Fi networks still support WPA-TKIP in 2016. These networks use TKIP as their group cipher, and hence are all susceptible to an attack we describe in this chapter.

The channel-based man-in-the-middle attack presented in this chapter has also proven to be useful in other situations. In particular, it is now also being used in our work *"Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys"*

to manipulate the 4-way handshake [187], and is used in *"Request and Conquer: Exposing Cross-Origin Resource Size"* to block selected background traffic sent by a client or AP [173].

3.1 Introduction

Wireless networks based on the 802.11 standard have had an enormous success, ranging from use in common households to large scale deployments in critical infrastructures. As new standards push the boundary of transmission speed and functionality, the capabilities of wireless chips have increased accordingly. This opens new possibilities where commodity devices can be used to implement state of the art attacks, previously thought only possible on expensive hardware such as Universal Software Radio Peripherals (USRPs). To demonstrate this we implement several low-layer attacks using off-the-shelf Wi-Fi dongles. In particular we modify the firmware of Atheros AR7010 and AR9271 chips. We conjecture that other devices, which also load user-modifiable firmware after power up, can execute similar attacks.

Our first modifications give the client an unfair share of the bandwidth. We use this to experimentally explore the behaviour of selfish stations. Based on this, to the best of our knowledge, we are the first to suggest that contending selfish stations will exploit the capture effect to increase throughput. This is the phenomenon where, in case of a collision, the frame with the strongest signal and lowest bitrate is received correctly [83]. Surprisingly, contending selfish stations end up lowering their bitrate to obtain a higher throughput. We also bypass systems designed to prevent selfish behaviour.

We continue by turning our dongle into a continuous and selective jammer. The former endlessly transmits noise, while the latter jams specific frames already in the air. Previously, selective jamming required either a costly USRP setup [34], or a Wi-Fi adapter which allows modifications to the microcode handling medium access and de/encoding operations [21]. Though it shows selective jamming is possible, the URSP setup is non-trivial and costs more than $3600. Additionally, being able to change medium access algorithms in commodity devices is rare in practice. We show that these jamming attacks can be implemented using off-the-shelf USB dongles (costing only 15$) without requiring low-level microcode access. It is surprising this is possible by using only the default (and limited) API of the wireless chip (i.e., without modifying its medium access algorithms).

With as goal to enable (or facilitate) attacks on higher-layer protocols, we also present a channel-based MitM attack against any type of encrypted network.

Our goal is not to decrypt traffic, but to be able to reliably intercept and manipulate it. Normally obtaining a MitM position in an encrypted network is not trivial, since the station and Access Point (AP) verify each other's MAC address. Therefore an attacker cannot set up a rogue access point with a different MAC address to subsequently intercept and forward all traffic. We demonstrate that cloning the AP on another channel bypasses the MAC address checks. Customizable firmware makes the implementation easier and more efficient, in particular when targeting multiple stations simultaneously.

Finally we use the MitM position to attack TKIP when used as a group cipher, and show how to decrease the execution time of the attack by targeting multiple stations simultaneously. Since all routers that support TKIP also use this cipher as their group cipher, more than half of all encrypted networks are vulnerable to our attack (see Chapter 2).

Our selective jammer, channel MitM, and TKIP attacks are available for download [175]. The unfair channel usage and continuous jammer can disrupt network activity without usable countermeasures being available. Therefore we do not release them. To summarize, our main contributions are:

- We study (contending) selfish stations using off-the-shelf Wi-Fi dongles, taking into account the physical layer capture affect, and bypass systems designed to detect selfish users.
- We show how continuous and selective jamming can be implemented using off-the-shelf Wi-Fi dongles.
- We present a channel-based Man-in-the-Middle attack against encrypted networks, enabling reliable interception and manipulation of all encrypted traffic.
- We attack TKIP when used as a group cipher, and additionally devise a technique to decrease the execution time by targeting multiple stations simultaneously.

The remainder of this chapter is organized as follows. Section 3.2 explains the relevant aspects of the 802.11 standard. In Section 3.3 we analyze, implement, and measure the impact of selfish stations. Section 3.4 shows how continuous and selective jamming can be implemented on commodity devices. A novel channel-based Man-in-the-Middle attack is presented in Sect. 3.5, and in Sect. 3.6 we use this to attack TKIP when used as a group cipher. Finally, we summarise related work in Sect. 3.7 and conclude in Sect. 3.8.

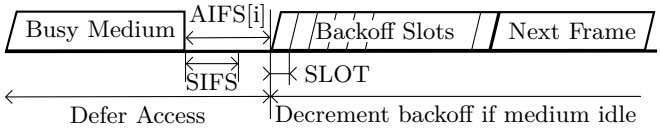

Figure 3.1: Visualisation of 802.11 EDCA timing relations.

3.2 The 802.11 Standard

In this section we present the basics of the 802.11 MAC and physical (PHY) layer [192], explain the TKIP encryption protocol, and introduce the `ath9k_htc` firmware.

3.2.1 Medium Access Control (MAC)

The MAC service exchanges MAC Protocol Data Units (MPDUs) using carrier sense multiple access with collision avoidance (CSMA/CA). A station wishing to transmit first monitors the medium for a given Inter Frame Space (IFS), using a carrier sense mechanism. If the medium is busy the station defers its transmission using a random backoff procedure. The original standard offered the Distributed Coordination Function (DCF) to control this process. The 802.11e amendment extends DCF with Quality of Service (QoS) enhancements, resulting in enhanced distributed channel access (EDCA). With EDCA four different Access Classes (AC) are defined, allowing the prioritization of traffic. For example, the access class of voice traffic has a higher priority than that of background traffic. We use the terms QoS channel and priority as synonyms of AC.

The time a station must wait before it can transmit depends on several parameters and inter frame spaces (see Fig. 3.1). The shortest is the Short Interframe Space (*SIFS*) and is the time between successfully receiving a frame and sending an acknowledgement (ACK). Other frames must wait longer than *SIFS*, giving ACKs the highest priority. When a station wants to transmit a data frame of access class AC it first monitors the medium for a period equal to

$$AIFS[AC] = SIFS + AIFSN[AC] \cdot SLOT\,, \tag{3.1}$$

where $SLOT$ is the duration of one slot interval, and with $AIFSN$ dependent on the access class. If the medium is idle during this time the station can transmit immediately. If the medium was busy the contention window $CW[AC]$ is set to $CW_{min}[AC]$ and the random backoff procedure is initiated. This

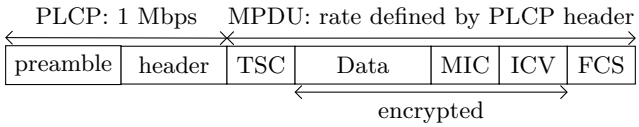

Figure 3.2: Simplified format of 802.11b TKIP frames.

procedure initializes the backoff counter to a value uniformly distributed over the interval $[0, CW[AC]]$. Once the medium has been idle for $AIFS[AC]$, the backoff counter is decremented for each slot where the medium is idle. When the counter is zero the station transmits the frame. Since AC is clear from context it will no longer be explicitly included. The values assigned to $AIFSN$, CW_{min}, and CW_{max} determine the priority of an access class, and are advertised by the AP in the EDCA parameter set element in beacons and probe responses. Beacon frames are periodically transmitted to advertise the presence of a network, while probe requests and responses are used to actively seek networks. The value of $SIFS$ and $SLOT$ depend on the physical layer being used (see Sect. 3.2.2).

If the Frame Check Sequence (FCS) at the receiver is correct it will wait $SIFS$ time and send an ACK. If the sender receives an ACK it resets CW to CW_{min} and enters a post-backoff stage by setting the backoff counter to a value uniformly distributed over the interval $[0, CW]$. When the FCS is wrong the frame is discarded. When the sender receives no ACK it retransmits the frame by updating CW to $2 \cdot CW + 1$ and repeating the backoff procedure. A frame is retransmitted at most 7 times, and CW is limited by CW_{max}.

Assuming all stations cooperate, analytical models of DCF show that it provides fairness [22]. Hence identical access classes in EDCA also provide this fairness. However, in practice stations can act selfishly and increase their throughput at the cost of others [129, 139]. In Sect. 3.3 we will study in detail how selfish stations can increase their throughput.

3.2.2 Physical Layer (PHY)

Wi-Fi can operate in the 2.4 and 5 GHz bands. A channel number defines the center frequency on which the radio operates. Senders can choose between multiple transmission rates: low rates are used for bad connections, and high rates for good connections. For backwards compatibility beacons and probes are generally sent using the lowest rate of 1 Mbps.

When an MPDU is transmitted a Physical Layer Convergence Protocol (PLCP) preamble and header is added (see Fig. 3.2). The preamble allows the receiver

to synchronize to the transmission. The header defines the modulation used for the MPDU, and contains the (remaining) length of the frame. We focus on 802.11g long preamble mode, where (slightly simplified) the overhead of the PLCP is 23 μs, *SIFS* is 10 μs, and *SLOT* is 9 μs [192, §18.3].

An important property of 802.11 radios is their susceptibility to the capture effect, the phenomenon where the strongest frame is received correctly in case of a collision. This goes against the common assumption that both frames are lost. More precisely, a collision results in one of three cases [83]:

1. both frames are lost;
2. if the stronger frame is received first it will be decoded correctly; or
3. if the stronger frame is received second, yet within the PLCP preamble of the first one, the second frame will be decoded correctly.

The capture effect has been observed on different chipsets [59, 83], occurs more frequently at low bitrates [89], and favours nearby stations [83]. In Section 3.3 we show that selfish stations can abuse it to contend with other stations.

3.2.3 Temporal Key Integrity Protocol

The Temporal Key Integrity Protocol (TKIP) is an improvement of the broken WEP protocol [164]. It was designed so old WEP-compatible hardware can support TKIP with only firmware updates. Nowadays TKIP is deprecated, and the more secure (AES-)CCMP is recommended. However, networks which support both TKIP and CCMP simultaneously, generally use TKIP as their group cipher. This is done for backward compatibility, so both old TKIP clients and newer CCMP clients can decrypt broadcast data. Since most routers by default allow both TKIP and CCMP, and thus use TKIP as their group cipher, TKIP is still widely used. For example, in Chapter 2 it has been shown that 66% of wireless networks in residential areas in Belgium use TKIP as their group cipher. Note that WPA1 and WPA2 are certifications handed out by the Wi-Fi Alliance. The difference between them is that WPA1 mandates TKIP support and optionally allows CCMP, while the reverse is true for WPA2. We use the term WPA to refer to both.

When a station connects to a secured network it begins by negotiating session keys using a 4-way handshake. This results in a pairwise transient key (PTK) and a group temporal key (GTK). These keys depend, among other things, on the MAC address of the station and AP. The PTK protects unicast traffic, while the GTK protects broadcast traffic. When using TKIP as the group cipher,

the GTK is used to derive a 128-bit encryption key, and a 64-bit Message Integrity Code (MIC) key. Note that a client transmitting a broadcast frame first sends it to the AP (as a unicast frame), after which the AP broadcasts it to all stations using the group cipher. All keys are renewed after a user defined interval, commonly set to 1 hour.

When a station transmits a TKIP protected frame, it first calculates the MIC over the data field using its 64-bit MIC key (see Fig. 3.2). Then it calculates a CRC over the data and MIC, called the Integrity Check Value (ICV). These fields are encrypted using a per packet key derived from the TKIP Sequence Counter (TSC) and 128-bit encryption key. The TSC is a replay counter which is incremented after successfully transmitting a frame. The receiver drops frames with an old TSC or bad ICV. However, if a frame has a valid TSC and ICV, but a wrong MIC, the TKIP countermeasures are activated:

- Clients send a MIC failure report to the AP. Additionally, when a client itself detects two failures within one minute, it will disconnect from the network.
- An AP silently logs the MIC failure. If two failures occur within one minute, it will halt all TKIP traffic for one minute. After this minute clients can reconnect.

Several weaknesses have been discovered in TKIP [69, 112, 116, 121, 125, 164, 168, 185]. The first known attack, and the one we built on, is the Beck and Tews attack [164]. It decrypts a packet byte by byte and requires that QoS is supported, which is the case for most modern devices [116, 185]. The attack works by guessing the last byte of an encrypted packet and then removing this byte. The ICV of the (encrypted) shortened packet can be corrected if the guess was correct. Because TKIP uses a unique TSC for each QoS channel, we can inject the corrected packet on a QoS channel with a lower TSC. If the guess was wrong, the ICV will be wrong, and the receiver drops the packet. However, if the guess was correct, the receiver will respond with a MIC failure. Hence the last byte of a packet can be found by trying all possible values and detecting a correct guess using the MIC failure report. This technique is then recursively applied to the shortened packet. To avoid the TKIP countermeasures at most one byte can be decrypted per minute. In the Beck and Tews attack this technique is used to decrypt an ARP packet. From the decrypted packet, the MIC key for that communication direction can be derived. This attack, and its variations, have previously only been designed to attack unicast traffic [69, 116, 121, 164, 168, 185].

For a more detailed background on TKIP, including practical attacks, we refer the reader to the previous chapter.

Listing 3.1: Code which sets EDCA parameters for a specific access class in the ath9k_htc driver.

```
1 int ath_htc_txq_update(...):
2     qi.tqi_aifs = qinfo->tqi_aifs;
3     qi.tqi_cwmin = qinfo->tqi_cwmin / 2; /* XXX */
4     qi.tqi_cwmax = qinfo->tqi_cwmax;
```

3.2.4 Atheros firmware implementation

The open source `ath9k_htc` firmware runs on AR7010 and AR9271 chips, and is sent to the device after power up. The AR7010 is a system-on-chip that generally uses PCIe to connect to an external AR9280 wireless chip, while the AR9271 is a single-chip solution with onboard wireless chip. The AR7010 supports both the 2.4 GHz and 5 GHz band, but can only use its built-in 2x2 MIMO antenna. On the other hand, the AR9271 can use an external antenna, but only supports the 2.4 GHz band.

The wireless chip in both devices is controlled using memory mapped registers. For example, the `AR_QTXDP` register contains a pointer to the transmit queue, and writing to the `AR_Q_TXE` register will start the transmit process.

3.3 Unfair Channel Usage

In this section we study (contending) selfish stations using off-the-shelf Wi-Fi dongles. This is done by implementing several strategies and measuring the resulting throughput. Finally we bypass system designed to detect selfish users.

3.3.1 Experimental setup

In order to implement selfish strategies we modified the `ath9k_htc` driver to get direct control over MAC layer parameters. While doing this we found that the original driver wrongly divided CW_{min} by two, while other parameters were properly configured (see Listing 3.1 line 3). This leads to an unfair advantage, and motivates other stations to act selfish as well. The driver was patched to properly set CW_{min}.

A Linksys WAG320N with firmware v1.00.08 is used as the AP. It is connected to a HP 8510p running Linux 3.7.2 using a Gigabit Ethernet connection. For the clients we created two identical VMWare Player instances, each having 2 GB

RAM and running Linux 3.7.2. One client uses a TL-WN722N placed 10 cm from the AP, while the other uses a AWUS036nha placed 1 meter from the AP (close distances reduce packet loss, making measurements more accurate). We refer to them as the close and far station, respectively. Reported results are over 10 runs of iperf 2.0.5 in UDP mode, using the maximum payload of an Ethernet frame (1 500 bytes minus 28 bytes for the IP and the UDP header), in the form bandwidth ± standard deviation. We use 802.11g, a channel width of 20 MHz, do not use the Request To Send (RTS) or Clear To Send (CTS) protection mechanism, and do not use encryption. We assume the AP to be trusted and only consider selfish clients.

3.3.2 One selfish station

We study the behaviour of a single selfish station in two phases. First when the selfish station is the only (sole) device connect to the network, and then when other fair stations are also connected to the network. For the first phase we let the close station be the only (sole) selfish station connected to the network. We implemented the following selfish strategies to determine whether they are effective or not:

1. Disabling backoff using the `AR_DMISC` register.
2. Setting *AIFSN* to zero in the `AR_DLCL_IFS` register.
3. Decreasing *SIFS* in the `AR_D_GBL_IFS_SIFS` register.

The result of these (incremental) strategies, including normal throughput, are shown in Fig. 3.3 under "Sole (selfish)". Only results where *SIFS* was lowered to 6 are shown (our conclusion is the same for other values). We now verify these results theoretically, by calculating the expected throughput, and comparing them with our empirical results. While doing this we assume there is no packet loss, and we assume the Maximum Transfer Unit (MTU) is 1 500 bytes. First, it takes 218 μs to send 1 472 bytes of UDP payload at 54 Mbps (the highest 802.11g bitrate), and 23 μs to send the long PLCP preamble and header. We have an overhead of a 34-byte MAC header, a 20-byte IP header, a 8-byte UDP header, and a 4-byte FCS. At 54 Mbps this results in a overhead of 33 μs for each frame encapsulating a 1 472 byte UDP payload. Since an ACK is 14 bytes and sent at 24 Mbps, it takes 38 μs to transmit when including the LPCP and *SIFS* timeout. Before sending the data frame, we wait on average $SIFS + (AIFSN + \frac{CW_{min}}{2}) \cdot SLOT$. Normally, when *SIFS* is 10 μs, $SLOT$ is 9 μs, $AIFSN$ is 3, and CW_{min} is 15, this is equal to 104.5 μs. Hence it takes on average 393 μs to send 1 472 bytes of data, resulting in a theoretical throughput

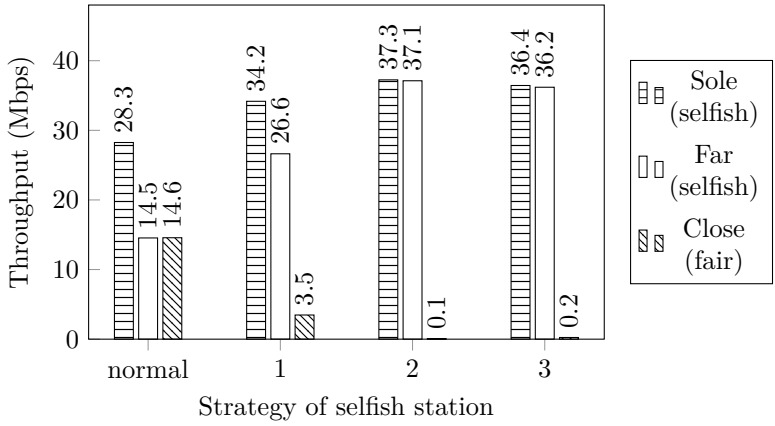

Figure 3.3: Throughput in case a selfish station is either the only station present (Sole), or in case there is another far station also present (Far and Close). The selfish station uses the strategies listed in Sect. 3.3.2. Standard deviation of all results were below 0.28 **Mbps.**

of 29.96 Mbps. For the three selfish strategies we get 36.18 Mbps, 39.45 Mbps, and 40.53 Mbps, respectively. Although our empirical results are slightly lower, they are still in line with these predicted throughput estimates. The empirical throughput is lower than predicted because beacons are also transmitted every 102.4 ms, and because not all packets arrive successfully. Notice that reducing *SIFS* may negatively impact frame delivery. We conjecture this is because it causes packets to arrive too quickly at the receiver. In other words, the receiver is not always ready to start processing the next frame, meaning some packets are being dropped. We conclude that a sole selfish station will disable backoff and set *AIFSN* to zero, but will not change *SIFS*.

The experiments were repeated when a fair station was also present. Since a malicious station may be unable to move arbitrarily close to the AP, we pick the far station as the selfish one. The results are shown in Fig. 3.3 under "Close (fair)" and "Far (selfish)". Again the optimal strategy of the selfish station is to keep the default value of *SIFS*, disable backoff, and set *AIFSN* to zero. We found this strategy works independent of the distances between stations. Hence we can conclude that a selfish station will always disable backoff and set *AIFSN* to zero, but will not lower *SIFS*, even when other stations are also present.

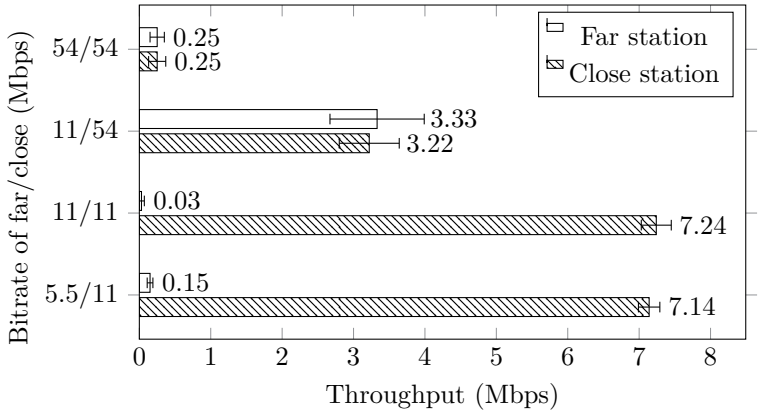

Figure 3.4: Impact of bitrate on contending selfish stations. Y-axis shows bitrate of far and close station, respectively. Error bars denote standard deviation.

3.3.3 Multiple selfish stations

We experimentally investigate the behaviour of contending selfish stations, by letting both the close and far station act selfishly. Because selfish stations generate so much traffic they hinder the arrival of beacons, it was necessary to modify the stations so they would not disconnect when too many beacons were lost. We found two strategies to increase throughput, both with the goal of exploiting the capture effect:

1. Reducing the bitrate of transmissions.
2. Increasing transmit power.

These strategies work because, in case of a collision, the receiver tends to decode the frame having the lowest bitrate and the highest signal strength. This also implies that selfish stations will immediately transmit frames, even if the medium is busy. In other words, they will disable carrier sense, which can be done using the `AR_DIAG_SW` register. To avoid the station from lowering its bitrate after a failed transmission, we force both stations to use a fixed bitrate in each experiment.

Our goal is now to determine which instantiation of the above two strategies the selfish stations will use. More specifically, we want to determine the bitrates that will be used, and whether an increased transmit power will be used in

combination with the selected bitrates. In a game theoretic framework, this would be similar to analyzing a non-cooperative game, where each player can make decisions independently. However, here we will take a more informal approach to analyze the behaviour of the selfish stations. In particular, we will empirically search for the point (equilibrium) at which neither station can further increase its throughput, and hence will not change its strategy.

First we performed experiments when contending selfish stations reduce their bitrate. We found that stations keep lowering their bitrate until this no longer provides any advantages (see Fig. 3.4). In particular, both stations start with a 54 Mbps bitrate. Due to collisions at high bitrates most frames are lost. Assuming the far station lowers its bitrate, while the close station still uses 54 Mbps, the bitrate resulting in the maximum throughput for the far station was 11 Mbps. In response the close station will maximize its own throughput by using 11 Mbps as well. This arms race continues until lowering the bitrate no longer provides any advantages. We conjecture that the final bitrates depend on the devices being used and on the environment. In our setting, the close station ends up using 11 Mbps while the far station uses 5.5 Mbps. At this point neither station can increase its throughput. In other words, a non-cooperative equilibrium has been reached. We conclude that contending selfish stations will lower their bitrate in order to get a higher throughput.

A higher transmit power can also increase throughput, though it is more sensitive to distances. If the close station is 10 cm from the AP, the far station was unable to noticeably increase its throughput by using a higher transmit power, independent of the bitrates being used. However, when placing the close station 70 cm from the AP the situation changes. Assuming both stations act selfishly and use 11 Mbps, the throughputs of the close and far station are 0.50 ± 0.52 Mbps and 2.10 ± 1.05 Mbps, respectively. When the far station uses an amplifier as external antenna, its throughput increases to 3.66 ± 2.08 Mbps while the close station gets 0.44 ± 0.28 Mbps. Hence selfish stations will use a higher transmit power, on top of lowering their bitrates, in an attempt to increase their throughput.

At higher distances both strategies still work. However, as the distance increases, it gets harder for a selfish station to beat other stations and increase its own throughput.

3.3.4 Defeating countermeasures

We now investigate mechanisms which are designed to detect (and prevent) selfish behaviour. Unfortunately we found that even advanced detection mechanisms such as DOMINO [139] have weaknesses. Though they can reliably

detect manipulations of interframe spaces and the backoff procedure, defending against stations which jam others appears difficult. The goal of jamming others is to increase their contention window (backoff counter), increasing the chance the cheater can access the channel. Frames of other stations can be jammed by a selective jammer (see Sect. 3.4.2).

An attacker only jams frames of other clients. It detects such frames based on the addresses in the MAC header. Hence detection mechanisms assume the header of frames remain valid, and use this to count the number of failed (jammed) transmissions of stations [139]. If a station has a significantly lower failed transmission count than all others, it is assumed to be jamming the other stations, and is punished (e.g. thrown off the network). The flaw in this technique is that the authenticity of jammed (corrupted) frames cannot be guaranteed. An attacker can forge jammed frames and fool the mechanism into thinking a station is jamming others. The targeted station is then wrongly punished.

We created a tool to forge jammed frames. In particular we used the `AR_DIAG_SW` register to force the wireless chip to append a bad Frame Check Sequence (FCS) to every transmitted frame. To target a victim we forge jammed frames which appear to come from all other stations. As a result the victim will appear to have a significantly lower number of failed transmissions compared to the others, and will be wrongly punished.

We conclude that existing systems tend to underestimate the power of attackers, and hope our work gives a better overview of both the capabilities and behaviour of attackers.

3.4 Jamming

In this section we show how to implement continuous and selective jamming on commodity Wi-Fi devices. To the best of our knowledge we are the first to accomplish this.

3.4.1 Continuous jamming

A continuous jammer transmits noise for an indefinite amount of time. The noise can consist of random energy pulses or data resembling the protocol being jammed. Depending of the device or network under attack, one of these options is typically preferred by the attacker. For instance, injecting random Wi-Fi

Table 3.1: Important memory-mapped registers and 802.11 radio functionality they control.

Register	Description
`AR_DIAG_SW`	Disable carrier sense, abort ongoing reception, append bad FCS
`AR_D_GBL_IFS_SIFS`	*SIFS*
`AR_D_GBL_IFS_SLOT`	*SLOT*
`AR_DLCL_IFS`	CW_{min}, CW_{max}, and *AIFSN*

frames might trigger warnings from intrusion detection systems, while random noise will be seen as non-Wi-Fi devices using the same frequency.

To turn a commodity Wi-Fi device into a continuous jammer we need to be able to carry out the following tasks:

1. Disable carrier sense.
2. Reset all interframe spaces and disable backoff.
3. Prevent the chip from waiting for an ACK.
4. Queue a large number of frames for transmission.

Achieving the first two points is explained in Sect. 3.3. The most important registers we used are summarized in Table 3.1.

To address the last two points we need to know how the wireless chip is instructed to transmit frames. The `ath9k_htc` firmware maintains a linked list of frames to be sent. Each frame is encapsulated by a transmit descriptor, which includes metadata describing how it should be transmitted. Among the metadata is the `HAL_TXDESC_NOACK` flag. When set, the wireless chip does not wait for an ACK and will not retransmit the frame. To queue an infinite number of frames we turn the linked list into one self-linked element.

We have implemented the continuous jammer by modifying the `ath9k_htc` firmware and created a tool to send commands to the firmware [175]. Since the AR7010 chip supports both the 2.4 and 5 GHz band, we can jam both bands on any channel. It is straightforward to turn our continuous jammer into a periodic jammer, which transmits only during certain intervals. An advantage of a periodic jammer is a lower power consumption.

We tested our implementation using a WNDA3200 and AWUS036nha. Both devices are able to jam any channel in the 2.4 GHz band, with the WNDA3200

Table 3.2: Devices and their wireless chips. Monitor mode is supported by all USB and Intel PCI devices.

Type	Wireless Chip	Device
USB	Atheros AR9271	Alfa AWUS036nha
	Atheros AR9271	TP-Link WN722N
	Atheros AR9280	Netgear WNDA3200
	Realtek RTL8187L	Alfa AWUS036h
	Ralink RT2720	Belkin F5D8053 v3
	Ralink RT3070	Digiflex aw-u150bb
PCI	Atheros AR5112	Wistron NeWeb CM9
	Broadcom bcm4334	Galaxy i9305
	Intel jc82535rde	Intel 4965AG
	Intel wg82541rde	Intel 5100AGN
Access Point (PCI)	Atheros AR2413	Sagem F@st 3464
	Broadcom bcm2050	WRT54g v2
	Broadcom bcm4322	Linksys WAG320N
	Broadcom bcm5356	Asus RT-N10

also able to jam channels in the 5 GHz band. All devices in Table 3.2 were susceptible to the attack: once the jammer was activated they all lost their connection to the AP. When monitoring the channel we observed that only the first frame injected by the jammer was visible. All devices in Table 3.2 supporting monitor mode displayed this behaviour. This highlights an important difference between our jammer and the unfair channel usage described in Sect. 3.3. Though the high load of a selfish node could at times cause a fair station to disconnect, all its traffic would be visible in monitor mode.

The effectiveness of the jammer depends on how it is blocking frames. They can be blocked by triggering the carrier sense mechanism of the transmitter (preventing it from sending frames) or by mangling the frame at the receiver. Generally it is easier to silence a transmitter, since less power is required to trigger the carrier sense mechanism compared to mangling incoming frames. To establish an upper bound on the effectiveness of our jammer we tested the maximum distance for which the AWUS036nha could silence a TL-WN722n. This was done with a clear line-of-sight between the victim and the jammer. Testing whether a device was silenced was done by letting it inject a frame, and using a second device to monitor whether it was transmitted. With an AWUS036nha as the jammer, it was effective up to roughly 80 meters. When using an amplifier as external antenna the range was extended to roughly 120 meters.

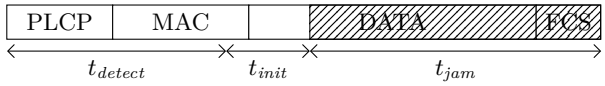

Figure 3.5: Timing requirements of selective jamming. After a frame header is detected and decoded, the jammer is initiated to jam the remaining content.

Cheap and readily available jammers for the 2.4 and 5 GHz bands can have a high impact since many devices operate in these bands (e.g., security webcams, baby monitors, etc).

3.4.2 Selective jamming

Selective jammers are arguably the most sophisticated and efficient. By targeting only selected frames an attacker can stealthy block specific frames from reaching their destination. In its simplest form a selective jammer emits noise whenever radio activity is detected. More sophisticated ones decode a prefix of a frame and then decide whether to jam the remaining content. In other words, they use higher-layer information to decide whether to jam it. The downside is that selective jammers can be difficult to implement, since they must adhere to strict timing requirements. This is illustrated in Fig. 3.5 where t_{detect} is the time it takes to detect and decode the frame header, t_{init} the time needed to start the jammer, and t_{jam} the time when the frame is being jammed. To successfully jam a frame $t_{detect} + t_{init}$ must be lower than the time it takes to transmit the frame.

To implement selective jamming on commodity devices we must be able to accomplish the following tasks:

1. Read the decoded header of a frame still in transit.
2. Abort receiving the current frame.
3. Immediately jam the channel.

Point 3 can be done by disabling carrier sense, setting all interframe spaces to zero, disabling the backoff procedure, and then injecting a dummy frame (see Sect. 3.4.1). Point 2 can be accomplished using the `AR_DIAG_SW` register.

There is no support to read the already decoded bytes of a frame still in transit. Nevertheless, one can still accomplish this if the wireless chip uses

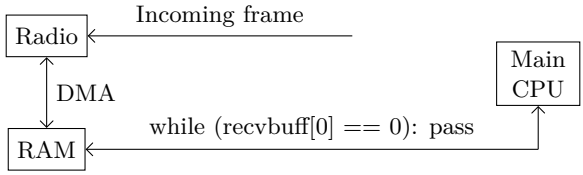

Figure 3.6: Visualization of how to use commodity devices to detect and process frames while they are still in the air.

Direct Memory Access (DMA) to write already decoded bytes to memory. In that case we can monitor the receive buffer using the main CPU (see Fig. 3.6). We first initialize the receive buffer with values that do not occur in valid 802.11 frames. Once these bytes are being modified, we know a frame is being received, and can read the already decoded bytes.

Implementation

We implemented the selective jammer by modifying the `ath9k_htc` firmware [175]. Currently it is designed to jam beacons and probe responses from a particular AP. The user can specify which SSID or MAC address to target and for how long to attack it. It is straightforward to modify our implementation to target different frames. In particular, the following fields can be used to target other frames: (1) MAC addresses in the header; (2) type and priority of frame; (3) sequence and fragment number. One can also use the first bytes of data in the packet, though this may make it difficult to jam the frame in time. Once we have decided to jam the frame, we inject a dummy frame transmitted at 1 Mbps to create a collision and mangle the frame. This will cause the FCS of the targeted frame to be invalid making the receiver drop the frame. To prevent the injected frame from being retransmitted we use the `HAL_TXDESC_NOACK` flag. When targeting unicast frames it is possible to inject an ACK immediately after jamming the frame. This prevents the sender from retransmitting the frame.

The selective jammer can also be modified to assure an attacker is the first to reply to certain frames. For example, an attacker can detect a probe request while it is still being transmitted, and immediately queue the transmission of a probe reply. To assure the reply is transmitted before any other station we disable backoff and set *AIFSN* to zero. A proof of concept of this attack has been implemented. The tool detects probe requests, jams them, and replies with a custom probe response. In Section 3.5 we use this to send probe responses with a modified channel number. Being able to reply faster than the legitimate

source is known to facilitate other attacks as well. For example, it can also be used to forge DNS replies in an (unencrypted) network.

We conjecture that, with sufficient efforts, all operations can be implemented on other commodity devices as well.

Experiments

We selectively jammed beacons with the victim at 70 cm from the AP and the jammer at 1 m (all devices were located on a single line). The victim was put in monitor mode to track how many beacons were malformed. In each experiment we carried out the attack for 2 minutes. When an AWUS036nha jams a WNDA3200, on average 52% of the beacons are malformed. However, when an AWUS036nha is jamming a TL-WN722N, 99% of the beacons are malformed. Hence the effectiveness depends on the device of the victim. More precisely, the type of antenna, radio chip, internal noise filters, etc. can all influence how vulnerable a victim is. The device of the attacker also plays an important role. As mentioned, when an AWUS036nha uses its default antenna to jam a WNDA3200, on average 52% of the beacons are malformed. But when connecting an amplifier to the AWUS036nha and then jamming the WNDA3200, 92% of all beacons are jammed. The location of the victim and attacker also plays an important role. When a WNDA3200 jams another WNDA3200 in the 2.4 GHz band, 51% of beacons were jammed. If we switch the location of the attacker and victim, all beacons were jammed. This is expected, since the signal strength of the AP is now lower for the victim, meaning it becomes easier for the attacker to overpower it. More precisely, the attacker must transmit a signal powerful enough to overcome capture effect, otherwise the victim still correctly receives the original frame. Finally we tested our jammer in the 5 GHz band by using two WNDA3200's. In this case all beacons were jammed.

With beacons sent at 1 Mbps in the 2.4 GHz band, both the AWUS036nha and WNDA3200 start mangling bytes at position 52 of the beacon. When beacons are sent at 6 Mbps in the 5 GHz band, the WNDA3200 starts mangling bytes at position 88. This is sufficient to jam beacons, probe requests, probe responses, and other packets of similar size. Currently the jammer is limited by the time it takes the wireless chip to write the first decoded bytes to RAM. In particular we observed that the wireless chip writes to RAM only after it has decoded the first 48 bytes. Further reverse engineering may reveal a technique to force to chip to write to RAM earlier, resulting in faster reaction times.

An interesting observation was made when selectively jamming an AR5112 chip. When the signal power of the jammer was ± 16 dB higher than that of the AP, the AR5112 returned a prefix of the beacon, followed by the dummy frame

injected by jammer. This was caused by the Message in Message (MiM) support of the AR5112. Devices that support MiM can resynchronise to a stronger frame while receiving a weaker frame, even if the stronger frame arrives after the preamble of the weaker frame [101]. Hence our selective jammer is an ideal tool to test whether a device supports MiM. For comparison, Lee et al. had to resort to more tedious experiments to show the AR5112 supports MiM [89].

Discussion

One limitation of our jammer is that it does not have access to the PLCP header of a frame still in transit. In particular this means we are unable to access the length field in the PLCP header, and thus do not know how long the frame is. Normally the `AR_DataLen` field in the receive descriptor contains the length of the frame. However, for frames still in transit, it contains the number of bytes written to RAM so far. We found no other data from which the length can be derived, though more analysis may overcome this obstacle.

Our cheap selective jammer is capable of deterministically creating collisions. Apart from being used as a jammer, it can also be used to test new collision detection techniques, experiment with Message in Message radios, etc. In short, a cheap and easily available selective jammer not only shows that jamming poses a more serious threat than previously thought, it also facilitates research experiments.

3.5 Channel-Based MitM

In this section we present a man-in-the-middle attack on WPA secured networks. Our goal is not to decrypt traffic, but to reliable intercept and manipulate it. In Section 3.6 we use this to attack TKIP when used as a group cipher.

3.5.1 Background

If the goal of an attacker is to reliably sniff and manipulate traffic, merely monitoring the channel is insufficient. Inevitably packets will be missed, and it is difficult to block and manipulate traffic. Though a selective jammer can block certain packets, it is not reliable enough (see Sect. 3.4.2). Another problem is that selective jammers inherently do not have access to the complete frame, giving an attacker limited information to decide whether to block it. All these limitations disappear when having a MitM position.

Establishing a MitM position in a WPA secured network is difficult due to the 4-way handshake. This is because the generated session keys depend on the MAC address of the AP and client. Therefore, if we use a rogue AP with a different MAC address, the handshake will fail. Using the same MAC address as the real AP is not possible since the client and AP would simply communicate with each other.

3.5.2 Intercepting encrypted traffic

To intercept all traffic we will clone the AP on a different channel and forward all traffic to the real AP. This requires two Wi-Fi dongles: one operating on the channel of the real AP, and one cloning the AP on a different channel. Because both Wi-Fi dongles are physically close to each other they can receive each others frames, even though they operate on different channels. Hence, blindly forwarding packets between both channels may create an infinite loop. To avoid this we keep track of recently forwarded frames using their sequence numbers, and only forward new frames. To advertise our rogue AP we transmit a beacon every 102.4 ms and reply to probe requests using custom probe responses.

Once our rouge AP is started, we want to force clients to connect to it. Unfortunately, injecting probe responses containing the channel of the rogue AP does not cause clients to switch channels. Selectively jamming all beacons and probe responses is also not reliable, as some frames will inevitably be missed. Additionally we found that if a client was recently connected to an AP, it will simply assume it is still present, and immediately send an authentication frame. This technique allows a mobile device to quickly reconnect to a network. The authentication message is only 34 bytes long and too short to be selectively jammed. Hence selective jamming cannot be used to force clients connect to the rogue AP. To avoid all these issues we continuously jam the channel of the real AP (see Sect. 3.4.1). This forces the clients to switch to our channel and connect to our rogue AP. Once the clients have switched, we can stop the jammer, and start forwarding frames between the clients and the real AP.

3.5.3 Implementation

We implemented the attack in a command-line tool [175]. It allows a user to specify which AP to clone, whether to jam the channel of the real AP until clients connect to our AP, and whether to write all intercepted traffic to file.

The firmware was modified so injected frames are retransmitted in case of failure, and so ACKs are generated when frames are sent to us. Retransmitting

frames is done by disabling the `HAL_TXDESC_NOACK` flag. Generating ACKs on the device cloning the AP is straightforward, as it only needs to listen on the MAC address of the AP. However, the device on the channel of the real AP must listen to the MAC addresses of all connected clients. To accomplish this we rely on virtual interface support, a hardware technology enabling a single device to listen on multiple MAC addresses. When a client is trying to connect to the rogue AP, our modified firmware simulates the addition of a new virtual interface.

We had to patch the driver and firmware to prevent modifications to forwarded frames. In particular we made the driver treat injected data frames as management frames. In the firmware we marked injected frames as CF-Poll frames to prevent the sequence number from being overwritten.

After establishing a MitM position an attacker will reliably capture all traffic. Additionally, packets can be blocked by not forwarding them. It is straightforward to update the tool so the attack can be executed even when the target is far away from the AP by forwarding frames over the internet.

3.5.4 Experiments

We performed several experiments to measure the reliability and impact of the attack. For the victim we used a Latitude E6500 running Linux 3.7.2, for the AP a Linksys WAG320N using firmware v1.00.08, and as attack machine a VMWare player instance having 2 GB RAM and running Linux 3.7.2 with our modified drivers and firmware. We used a TL-WN722N and WNDA3200 to intercept and forward traffic, and an AWUS036nha as the jammer. The AP was configured to operate in 802.11g mode.

When using continuous jamming to force clients to connect to our rogue AP, we consistently established a MitM position. Even clients already connected to the real AP switched to the rogue AP. We then measured the impact on latency, bandwidth, and web page loading times. When connected to the real AP the victim had a latency of 3.82 ± 10.4 ms. This increased to 7.45 ± 12.9 ms when connected to our rogue AP. Hence latency is doubled, which is expected since every packet must be forwarded by our rogue AP, and is thus transmitted twice. To test the impact on throughput we let the victim download a 100 MB file. Under normal conditions this takes place at 18.6 Mbps. When being attacked the speed is lowered to 8 Mbps. Finally we tested the impact on page load times when surfing the web. Under normal conditions a page was loaded in 0.76 ± 0.07 ms, which increased to 0.83 ± 0.08 ms when under attack. Since this is only a 9% slowdown users are unlikely to notice this and will not realise they are under attack.

These results can be optimized by implementing a rate adaptation algorithm when forwarding packets, and by using unfair MAC parameters as described in Sect. 3.3.

3.5.5 Countermeasures

There are legitimate reasons for an AP to change channel, for example to switch to a less occupied one. In the 5 GHz band, switching channels is even required to avoid interference with radars [192, §4.5.5.3]. Hence storing the channel of an AP you previously connected to is not feasible.

The attack can be detected by including the channel of the AP in the 4-way handshake. Unfortunately this requires a change in the existing protocol, making this difficult to realise in practise. Ideally the impact of the attack is reduced by using a secure encryption protocol such as CCMP.

3.6 TKIP as a Group Cipher

In this section we attack TKIP when used as a group cipher, which is the default security setting for most routers.

3.6.1 Attack details

Previously TKIP was only investigated when used to protect unicast traffic (see Sect. 3.2.3). Our goal is to show that TKIP can also be attacked when used as a group cipher, i.e., when used to protect broadcast and multicast frames.

Directly applying the Beck and Tews attacks on broadcast packets fails when multiple clients are connected to the AP. In that case all clients will simultaneously send a MIC failure report (instead of only the targeted client as in the unicast scenario). Hence the AP immediately starts the TKIP countermeasures. Recall that the countermeasures disable all TKIP traffic for one minute, after which a new GTK is generated and clients can reconnect. Thus we would only be able to decrypt one byte. Our selective jammer cannot be used to block the MIC failure reports, because adversaries detect these reports based on their unique length. More troublesome, MIC failures are generally sent at high bitrates, meaning even more advanced selective jammers are unable to reliably jam them. Instead we use the MitM attack from Sect. 3.5, allowing us to block MIC failure reports by not forwarding them. Note that we

must still assure that individual clients do not send more than one MIC failure report every minute. Otherwise the client will disconnect from the network, even if the AP did not start the TKIP countermeasures.

Once a MitM position has been achieved we wait until a small broadcast packet is transmitted. We prefer small packets as these take less time to decrypt. Since ARP requests are small and sent when a client (re)connects to a network, they are an ideal candidate. After capturing an ARP request we decrypt its ICV and MIC using the Beck and Tews method, and guess the remaining content. Since we do not forward MIC failures to the AP, the TKIP countermeasures will not be actived when multiple clients send a MIC failure report. Once the packet has been decrypted we learn the keystream corresponding to the sequence counter (TSC) of the ARP request, and we can derive the MIC key for broadcast traffic [164]. With this we can inject 3 to 7 small packets to any client connected to the AP (the amount depends on the number of supported QoS channels). Furthermore, existing attacks relying on knowing the MIC key can be modified to work on broadcast packets. For example, by modifying the attacks presented in Chapter 2, we can abuse fragmentation to inject an arbitrary amount broadcast of packets, and we can efficiently decrypt arbitrary broadcast packets.

We continue by speeding up the attack. There are several techniques to accomplish this. First, if we do not detect a MIC failure after trying all possible 256 values, we know the target did not receive the packet with the correct guess. If it did, it would retransmit the MIC failure until we responded with an ACK. Hence we can immediately retry guessing all values without fear of generating two MIC failures within one minute. In contrast, previous attacks on TKIP had to wait one minute because the attacker might have missed the MIC failure report while the AP did receive it.

If we are able to send broadcast packets to only one client we can further speed up the attack. We start the attack by sending guesses to one particular client. Once it sends a MIC failure report, we can immediately continue the attack by targeting another client. Each client will only see one MIC failure every minute, and the AP will see none, hence the TKIP countermeasures are never activated. We keep targeting different clients while assuring that an individual client will never send more than one MIC failure report every minute. The difficulty is in sending a broadcast packet to a specific client. Though it is possible by cloning the AP on multiple channels and spreading the clients out over these channels, such an approach is cumbersome and requires multiple Wi-Fi devices. Instead we rely on the observation that most devices do not process their own broadcast packets. That is, most devices will drop broadcast packets with as source address their own MAC address. Assuming we are attacking two clients simultaneously, we can target one client by using the source address of the

other client. Finally we do not have to wait one minute when guessing the last remaining bytes. Though a client may send more than one MIC failure report within one minute, and therefore disconnect from the network, the AP will not see these MIC failures. Hence the AP will not activate its countermeasures.

3.6.2 Implementation and experiments

We implemented the attack by extending the MitM implementation (see Sect. 3.5). To attack two clients we rely on the fact that a client will not process broadcast packets with as source its own MAC address. Once the clients are connected to our rogue AP, and completed the 4-way handshake, we wait for an ARP request. The ARP request is decrypted byte by byte, and finally the MIC key is derived.

Our experimental setup is the same as in Sect. 3.5.4, except that we now also use a Samsung Galaxy i9305 running Android 4.3 as a second victim. During the experiment we first let the two victims connect to the real AP, after which we started the attack. The execution time over 10 runs was 7.3 ± 0.6 minutes. This is the time from launching the tool to decrypting the APR reply and deriving the MIC key. Note that the execution time can be further reduced by targeting more stations simultaneously. Program output and network traces of the attack are available for download [175].

As a comparison we measured the execution time of the original Beck and Tews attack. The attacker used a TL-WN722N and the victim an AWUS036h. They were placed close to each other to reduce packet loss, an hence represents an optimal case. We used the tkiptun-ng tool from aircrack-ng. The execution time over 10 runs was 14.6 ± 1.1 minutes, more than twice the time of our attack.

3.6.3 Countermeasures

To prevent the interception of MIC failure reports, APs should securely acknowledge them. If the client does not receive the acknowledgement within time, it may be under attack, and should activate its TKIP countermeasures. This prevents an attacker from blocking MIC failure reports and attacking multiple stations simultaneously.

The attack can be mitigated by using a short rekeying timeout of 2 minutes or less. Ideally the attack is prevented by using a more secure encryption protocol such as CCMP.

3.7 Related Work

Atheros drivers are popular for low-layer control over experiments, relevant examples are [59, 89, 139]. Custom firmware has also been used for low-layer control [19, 21, 68].

A significant amount of research on the behaviour of selfish stations rely only on analytic models and simulations [32, 135, 138]. Raya et al. perform experimental tests when a single node manipulates the contention window of the backoff procedure [139]. Pelechrinis et al. also perform experiments, but only consider stations which modify their Clear Channel Assessment (CCA) threshold [129]. However, an attacker can easily change additional parameters. We are not aware of any works doing a detailed experimental study on the behaviour of selfish stations, where multiple strategies are tested, and the capture effect is taken into account. The capture has been observed for 802.11 on Prism and Atheros chipsets [59, 83]. In [59] the authors attempt to reduce the unfairness caused by this effect. Lee et al. perform a detailed study on when the capture effect takes places, and found that the Atheros AR5112 not only exhibits capture effect but also supports Message in Message mode [89].

Bayraktaroglu et al. required a USRP to continuously jam the 2.4 GHz band [17]. Kim et al. modify the MadWifi driver of the Atheros 5212 to continuously emit meaningless frames in the 2.4 GHz band [80]. Noubir et al. used one USRP1 and two RFX2400 boards to build a selective jammer [119]. Unfortunately the reaction time of their setup was too slow to reliably jam frames [119, §6.1.4]. Cassola et al. required two USRP2s and two RFX2400 boards, totalling more than $3600, to selectively jam frames in the 2.4 GHz band [34]. Their jammer starts mangling frames sent using 1 Mbps at byte 38, while our jammer does this at byte 52. However, our dongle costs only $15, can jam both frequency bands, and is easier in use due to its smaller size. In concurrent and independent work, Berger et al. used custom microcode to have control of medium access, encoding, and decoding operations, and used this to implement selective jamming in the 2.4 GHz band [21]. The advantage of our approach is that we do not require such low-level control over medium access operations performed in the chip. Continuous and selective jammers have also been created to target other wireless protocols [191, 195]. Our selective Wi-Fi jammer is unique because it is both cheap and portable. Jamming can also be used as a defensive mechanism [21, 32, 191].

Beck and Tews found the first practically exploitable vulnerability in TKIP, which required QoS enhancements to be enabled [164]. Ohigashi and Morri used a MitM position to execute the attack even if QoS was not enabled [121]. In particular they assumed the client was not in range of the AP, and that the

attacker acted as a repeater by forwarding all frames between the client and AP. In contrast, our MitM attack does not require that the client and AP are out of range. Over the years additional improvements of the Beck and Tews attack have been found [69, 116, 168, 185]. Several more theoretical attacks on TKIP have also been published [112, 125, 156]. Finally, the closest related work to our channel-based MitM attack we are aware of is the AirJack tool [94]. It clones an unencrypted network on a different channel and forwards all traffic over Ethernet. Hence it cannot clone encrypted networks, whereas our attack can.

3.8 Chapter Conclusion

We were able to implement several low-layer attacks on Wi-Fi using open source Atheros firmware. This is surprising, since we only have access to a limited API to control the radio (e.g., we cannot transmit arbitrary signals, do not have access to raw signal data, cannot modify medium access algorithms, etc). Additionally we bypassed systems designed to prevent some of these attacks, indicating that previous works have underestimated the capabilities of attackers. We also showed that our low-layer attacks facilitate attacks on higher-layer protocols, by attacking TKIP when used as a group cipher. Since TKIP is used significantly more as a group cipher than as a unicast cipher, this demonstrates that weaknesses in TKIP are still of high practical value.

Finally, the selective jammer, channel MitM, and TKIP attacks are available for download [175]. We hope these results aid in the creation of better countermeasures, and motivate people to only use the more secure (AES)-CCMP.

Chapter 4

Breaking RC4 in WPA-TKIP and TLS

> "Having everything go your way is the road to disillusionment. Knowing the difference between what's impossible and what's possible. . . "
>
> — *Caitlin, Pokémon HeartGold*

Abstract

We present new biases in RC4, break the Wi-Fi Protected Access Temporal Key Integrity Protocol (WPA-TKIP), and design a practical plaintext recovery attack against the Transport Layer Security (TLS) protocol. To empirically find new biases in the RC4 keystream we use statistical hypothesis tests. This reveals many new biases in the initial keystream bytes, as well as several new long-term biases. Our fixed-plaintext recovery algorithms are capable of using multiple types of biases, and return a list of plaintext candidates in decreasing likelihood.

To break WPA-TKIP we introduce a method to generate a large number of identical packets. This packet is decrypted by generating its plaintext candidate list, and using redundant packet structure to prune bad candidates. From the decrypted packet we derive the TKIP MIC key, which can be used to inject and decrypt packets. In practice the attack can be executed within an hour. We

also attack TLS as used by HTTPS, where we show how to decrypt a secure cookie with a success rate of 94% using $9 \cdot 2^{27}$ ciphertexts. This is done by injecting known data around the cookie, abusing this using Mantin's $ABSAB$ bias, and brute-forcing the cookie by traversing the plaintext candidates. Using our traffic generation technique, we are able to execute the attack in merely 75 hours.

Preamble

Parts of this chapter were previously published as:

> M. Vanhoef and F. Piessens. "All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS". in: *Proceedings of the 24th USENIX Security Symposium (USENIX Security '15)*. USENIX Association, Aug. 2015, pp. 97–112

The main research effort of this work was lead by Mathy Vanhoef, under the supervision of Frank Piessens.

This work won the best student paper award at USENIX Security 2015. In combination with other recent attacks against RC4 [5, 60, 98], and due to the IETF publishing RFC 7465 which prohibits the usage of RC4 in TLS [131], this motivated major browser vendors to disable support of RC4. In particular, Google Chrome [86] and Mozilla Firefox [13] dropped support for RC4 in January 2016. While Microsoft planned to drop support of RC4 in April 2016, customer feedback made them prolong the support of RC4 [110]. Consequently, as of June 2016, Internet Explorer and Edge still support RC4.

After the publication of this work, the usage of RC4 in TLS connections has steadily kept dropping. While in February 2015 around 30% of all TLS connections used RC4, in July 2015 this dropped to 13%, and in April 2016 this further decreased to 3% [73].

To highlight the weaknesses in RC4, and encourage people to stop using it, the website `www.rc4nomore.com` was created. It explains the results of our work in layman's terms, and includes a video where we demonstrate how to decrypt a HTTP cookie protected using TLS and RC4. This work paid off: our website got picked up by Ars Technica [65], by Bruce Schneier on his personal blog [151], by The Register [36], by Golem News [61], etc. We believe this helped to sped up the removal of RC4. On this website we also published several datasets we generated, as well as the source code of some statistical tests we used. In addition to these public datasets, researchers from Bonn university requested our per-TSC statistics used in the attack against WPA-TKIP, in order

to replicate parts of this work. This dataset took 10 CPU years to generate, but was not made public due to its large 8 GiB size.

In contrast to the original publication, this chapter contains a more detailed analysis of new long-term biases that we discovered in the RC4 keystream. This additional work encompassed the generation of more detailed datasets, the analysis of these datasets, and a first step in the theoretical analysis of their cause. More concretely, Section 4.3.4 has been updated, and contains new results and findings.

4.1 Introduction

RC4 is (still) one of the most widely used stream ciphers. Arguably its most well known usage is in SSL and WEP, and in their successors TLS [43] and WPA-TKIP [192]. In particular it was heavily used after several attacks against CBC-mode encryption schemes in TLS were published, such as BEAST [46], the padding oracle attack [33], and Lucky 13 [6]. As a mitigation RC4 was recommended. Because of this, in 2013 around 50% of all TLS connections were using RC4 [5], and even in 2015 roughly 30% of all TLS connections still used RC4 [183]. This motivated the search for new attacks, relevant examples being [5, 60, 67, 75, 124, 125]. Of special interest to us is the attack proposed by AlFardan et al., where roughly $13 \cdot 2^{30}$ ciphertexts are required to decrypt a cookie sent over HTTPS [5]. This corresponds to about 2 000 hours of data in their setup, hence the attack is considered close to being practical. Our goal is to see how far these attacks can be pushed by exploring three areas. First, we search for new biases in the keystream. Second, we improve fixed-plaintext recovery algorithms. Third, we demonstrate techniques to perform our attacks in practice.

First we empirically search for biases in the keystream. This is done by generating a large amount of keystream, and storing statistics about them in several datasets. The resulting datasets are then analyzed using statistical hypothesis tests. Our null hypothesis is that a keystream byte is uniformly distributed, or that two bytes are independent. Rejecting the null hypothesis is equivalent to detecting a bias. Compared to manually inspecting graphs, this allows for a more large-scale analysis. With this approach we found many new biases in the initial keystream bytes, as well as several new long-term biases.

We break WPA-TKIP by decrypting a complete packet using RC4 biases and deriving the TKIP MIC key. This key can be used to inject and decrypt packets (see Chapter 2). In particular we modify the plaintext recovery attack of Paterson et al. [124, 125], which abused biases induced by the weak per-packet

key construction of WPA-TKIP, to return a list of candidates in decreasing likelihood. Bad candidates are detected and pruned based on the (decrypted) CRC of the packet. This increases the success rate of simultaneously decrypting all unknown bytes. We achieve practicality using a novel method to rapidly inject identical packets into a network. In practice the attack can be executed within an hour.

We also attack RC4 as used in TLS and HTTPS, where we decrypt a secure cookie in realistic conditions. This is done by combining the $AB\mathcal{S}AB$ and Fluhrer-McGrew biases using variants of attacks by Isobe et al. and AlFardan et al. [5, 75]. Our technique can also be extended to include other biases. To abuse Mantin's $AB\mathcal{S}AB$ bias we inject known plaintext around the cookie, and exploit this to calculate Bayesian plaintext likelihoods over the unknown cookie. We then generate a list of (cookie) candidates in decreasing likelihood, and use this to brute-force the cookie in negligible time. The algorithm to generate candidates differs from the WPA-TKIP one due to the reliance on double-byte instead of single-byte likelihoods. All combined, we need $9 \cdot 2^{27}$ encryptions of a cookie to decrypt it with a success rate of 94%. Finally we show how to make a victim generate this amount within only 75 hours, and execute the attack in practice.

To summarize, our main contributions are:

- We use statistical tests to empirically detect biases in the keystream, revealing large sets of new biases.
- We design plaintext recovery algorithms capable of using multiple types of biases, which return a list of plaintext candidates in decreasing likelihood.
- We demonstrate practical exploitation techniques to break RC4 in both WPA-TKIP and TLS.

The remainder of this chapter is organized as follows. Section 4.2 gives a background on RC4, TKIP, and TLS. In Section 4.3 we introduce hypothesis tests and report new biases. Plaintext recovery techniques are given in Sect. 4.4. Practical attacks on TKIP and TLS are presented in Sect. 4.5 and Sect. 4.6, respectively. Finally, we summarize related work in Sect. 4.7 and conclude in Sect. 4.8.

4.2 Background

In this section we introduce RC4 and its usage in TLS and WPA-TKIP.

Listing (4.1) RC4 Key Scheduling Algorithm (KSA)

```
1 j, S = 0, range(256)
2 for i in range(256):
3    j += S[i] + key[i % len(key)]
4    swap(S[i], S[j])
5 return S
```

Listing (4.2) RC4 Keystream Generation Algorithm (PRGA)

```
1 S, i, j = KSA(key), 0, 0
2 while True:
3    i += 1
4    j += S[i]
5    swap(S[i], S[j])
6    yield S[S[i] + S[j]]
```

Figure 4.1: Implementation of RC4 in Python-like pseudo-code. All additions are performed modulo 256.

4.2.1 The RC4 algorithm

The RC4 algorithm is intriguingly short and known to be very fast in software. It consists of a Key Scheduling Algorithm (KSA) and a Pseudo-Random Generation Algorithm (PRGA), which are both shown in Fig. 4.1. The state consists of a permutation $\mathcal{S}$ of the set $\{0, \ldots, 255\}$, a public counter i, and a private index j. The KSA takes as input a variable-length key and initializes $\mathcal{S}$. At each round $r = 1, 2, \ldots$ of the PRGA, the yield statement outputs a keystream byte Z_r. All additions are performed modulo 256. A plaintext byte P_r is encrypted to ciphertext byte C_r using $C_r = P_r \oplus Z_r$.

Short-term biases

Several biases have been found in the initial RC4 keystream bytes. We call these short-term biases. The most significant one was found by Mantin and Shamir. They showed that the second keystream byte is twice as likely to be zero compared to uniform [100]. Or more formally that $\Pr[Z_2 = 0] \approx 2 \cdot 2^{-8}$, where the probability is over the random choice of the key. Because zero occurs more often than expected, we call this a positive bias. Similarly, a value occurring less often than expected is called a negative bias. This result was extended by Maitra et al. [97] and further refined by Sen Gupta et al. [154] to show that there is a bias towards zero for most initial keystream bytes. Sen Gupta et al. also found key-length dependent biases: if ℓ is the key length, keystream byte Z_ℓ has a positive bias towards $256 - \ell$ [154]. AlFardan et al. showed that all initial 256 keystream bytes are biased by empirically estimating their probabilities when 16-byte keys are used [5]. While doing this they found additional strong biases, an example being the bias towards value r for all positions $1 \leq r \leq 256$. This bias was also independently discovered by Isobe et al. [75].

Table 4.1: Generalized Fluhrer-McGrew (FM) biases. Here i is the public counter in the PRGA and r the position of the first byte of the digraph. Probabilities for long-term biases are shown (for short-term biases see Fig. 4.4).

Digraph value	Condition	Probability
(0,0)	$i = 1$	$2^{-16}(1 + 2^{-7})$
(0,0)	$i \neq 1, 255$	$2^{-16}(1 + 2^{-8})$
(0,1)	$i \neq 0, 1$	$2^{-16}(1 + 2^{-8})$
(0,$i + 1$)	$i \neq 0, 255$	$2^{-16}(1 - 2^{-8})$
($i + 1$,255)	$i \neq 254 \wedge r \neq 1$	$2^{-16}(1 + 2^{-8})$
(129,129)	$i = 2 \wedge r \neq 2$	$2^{-16}(1 + 2^{-8})$
(255,$i + 1$)	$i \neq 1, 254$	$2^{-16}(1 + 2^{-8})$
(255,$i + 2$)	$i \in [1, 252] \wedge r \neq 2$	$2^{-16}(1 + 2^{-8})$
(255,0)	$i = 254$	$2^{-16}(1 + 2^{-8})$
(255,1)	$i = 255$	$2^{-16}(1 + 2^{-8})$
(255,2)	$i = 0, 1$	$2^{-16}(1 + 2^{-8})$
(255,255)	$i \neq 254 \wedge r \neq 5$	$2^{-16}(1 - 2^{-8})$

The bias $\Pr[Z_1 = Z_2] = 2^{-8}(1-2^{-8})$ was found by Paul and Preneel [127]. Isobe et al. refined this result for the value zero to $\Pr[Z_1 = Z_2 = 0] \approx 3 \cdot 2^{-16}$ [75]. In [75] the authors searched for biases of similar strength between initial bytes, but did not find additional ones. However, we did manage to find new ones (see Sect. 4.3.3).

Long-term biases

In contrast to short-term biases, which occur only in the initial keystream bytes, there are also biases that keep occurring throughout the whole keystream. We call these long-term biases. For example, Fluhrer and McGrew (FM) found that the probability of certain digraphs, i.e., consecutive keystream bytes (Z_r, Z_{r+1}), deviate from uniform throughout the whole keystream [56]. These biases depend on the public counter i of the PRGA, and are listed in Table 4.1 (ignoring the condition on r for now). In their analysis, Fluhrer and McGrew assumed that the internal state of the RC4 algorithm was uniformly random. This assumption is only true after a few rounds of the PRGA [56, 111, 154]. Consequently these biases were generally not expected to be present in the initial keystream bytes. However, in Sect. 4.3.3 we show that most of these biases do occur in the initial keystream bytes, albeit with different probabilities than their long-term variants.

Another long-term bias was found by Mantin [99]. He discovered a bias towards the pattern $AB\mathcal{S}AB$, where A and B represent byte values, and $\mathcal{S}$ a short sequence of bytes called the gap. With the length of the gap $\mathcal{S}$ denoted by g, the bias can be written as:

$$\Pr[(Z_r, Z_{r+1}) = (Z_{r+g+2}, Z_{r+g+3})] = 2^{-16}(1 + 2^{-8}\mathrm{e}^{\frac{-4-8g}{256}}) . \qquad (4.1)$$

Hence the bigger the gap, the weaker the bias. Bricout et al. refined this result, and showed that for certain values of Z_r and Z_{r+1}, the bias is non-existent, or stronger than predicted by (4.1) [29]. Finally, Sen Gupta et al. found the long-term bias [154]

$$\Pr[(Z_{w256}, Z_{w256+2}) = (0, 0)] = 2^{-16}(1 + 2^{-8}) ,$$

where $w \geq 1$. We discovered that a bias towards $(128, 0)$ is also present at these positions (see Sect. 4.3.4).

4.2.2 TKIP cryptographic encapsulation

The design goal of WPA-TKIP was for it to be a temporary replacement of WEP [192, §11.4.2]. While it is being phased out by the Wi-Fi Alliance, our survey in Chapter 2 showed its usage is still widespread in 2016. That is, out of 7 586 networks, we found that 59% of all protected networks allow TKIP, with 3% still exclusively supporting TKIP.

Our attack on TKIP relies on two elements of the protocol: its weak Message Integrity Check (MIC) [164, 185], and its faulty per-packet key construction [5, 67, 124, 125]. We briefly introduce both aspects, assuming a 512-bit Pairwise Transient Key (PTK) has already been negotiated between the Access Point (AP) and client. From this PTK a 128-bit temporal encryption key (TK) and two 64-bit Message Integrity Check (MIC) keys are derived. The first MIC key is used for AP-to-client communication, and the second for the reverse direction. Some works claim that the PTK, and its derived keys, are renewed after a user-defined interval, commonly set to 1 hour [164, 185]. However, we found that generally only the Groupwise Transient Key (GTK) is periodically renewed. Interestingly, our attack can be executed within an hour, so even networks which renew the PTK every hour can be attacked.

When the client wants to transmit a payload, it first calculates a MIC value using the appropriate MIC key and the Michael algorithm (see Fig. 4.2). Unfortunately Michael is straightforward to invert: given plaintext data and its MIC value, we can efficiently derive the MIC key [164]. After appending the MIC value, a CRC checksum called the Integrity Check Value (ICV) is also appended. The

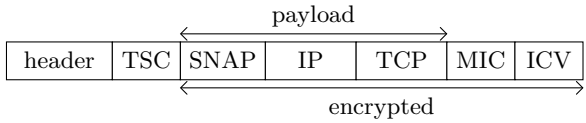

Figure 4.2: Simplified TKIP frame with a TCP payload.

resulting packet, including MAC header and example TCP payload, is shown in Fig. 4.2. The payload, MIC, and ICV are encrypted using RC4 with a per-packet key. This key is calculated by a mixing function that takes as input the TK, the TKIP sequence counter (TSC), and the transmitter MAC address (TA). We write this as $\mathrm{K} = \mathrm{KM}(\mathrm{TA}, \mathrm{TK}, \mathrm{TSC})$. The TSC is a 6-byte counter that is incremented after transmitting a packet, and is included unencrypted in the MAC header. In practice the output of KM can be modelled as uniformly random [5, 125]. In an attempt to avoid weak-key attacks that broke WEP [57], the first three bytes of K are set to [192, §11.4.2.1.1]:

$$\mathrm{K}_0 = \mathrm{TSC}_1 \qquad \mathrm{K}_1 = (\mathrm{TSC}_1 \mid 0\mathrm{x}20) \,\&\, 0\mathrm{x}7\mathrm{f} \qquad \mathrm{K}_2 = \mathrm{TSC}_0$$

Here, TSC_0 and TSC_1 are the two least significant bytes of the TSC. Since the TSC is public, so are the first three bytes of `K`. Both formally and using simulations, it has been shown this actually weakens security [5, 67, 124, 125].

4.2.3 The TLS record protocol

We focus on the TLS record protocol when RC4 is selected as the symmetric cipher [43]. In particular we assume the handshake phase is completed, and a 48-byte TLS master secret has been negotiated.

To send an encrypted payload, a TLS record of type application data is created. It contains the protocol version, length of the encrypted content, the payload itself, and finally an HMAC [85]. The resulting layout is shown in Fig. 4.3. The HMAC is computed over the header, a sequence number incremented for each transmitted record, and the plaintext payload. Both the payload and HMAC are encrypted. At the start of a connection, RC4 is initialized with a key derived from the TLS master secret. This key can be modelled as being uniformly random [5]. None of the initial keystream bytes are discarded.

In the context of HTTPS, one TLS connection can be used to handle multiple HTTP requests. This is called a persistent connection. Slightly simplified, a server indicates support for this by setting the HTTP `Connection` header to `keep-alive`. This implies RC4 is initialized only once to send all HTTP

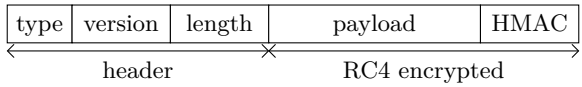

Figure 4.3: TLS Record structure when using RC4.

requests, allowing the usage of long-term biases in attacks. Finally, cookies can be marked as being `secure`, assuring they are transmitted only over a TLS connection.

4.3 Empirically Finding New Biases

In this section we explain how to empirically yet soundly detect biases. While we discovered many biases, we will not use them in our attacks. This simplifies the description of the attacks. And, while using the new biases may improve our attacks, using existing ones already sufficed to significantly improve upon existing attacks. Hence our focus will mainly be on the most intriguing new biases.

4.3.1 Soundly detecting biases

In order to empirically detect new biases, we rely on hypothesis tests. That is, we generate keystream statistics over random RC4 keys, and use statistical tests to uncover deviations from uniform. This allows for a large-scale and automated analysis. To detect single-byte biases, our null hypothesis is that the keystream byte values are uniformly distributed. To detect biases between two bytes, one may be tempted to use as null hypothesis that the pair is uniformly distributed. However, this falls short if there are already single-byte biases present. In this case single-byte biases imply that the pair is also biased, while both bytes may in fact be independent. Hence, to detect double-byte biases, our null hypothesis is that they are independent. With this test, we even detected pairs that are actually more uniform than expected. Rejecting the null hypothesis is now the same as detecting a bias.

To test whether values are uniformly distributed, we use a chi-squared goodness-of-fit test. A naive approach to test whether two bytes are independent, is using a chi-squared independence test. Although this would work, it is not ideal when only a few biases (outliers) are present. Moreover, based on previous work we expect that only a few values between keystream bytes show a clear dependency on each other [16, 56, 75, 99, 154]. Taking the Fluhrer-McGrew biases as an

example, at any position at most 8 out of a total 65 536 value pairs show a clear bias [56]. When expecting only a few outliers, the M-test of Fuchs and Kenett can be asymptotically more powerful than the chi-squared test [58]. Hence we use the M-test to detect dependencies between keystream bytes. To determine which values are biased between dependent bytes, we perform proportion tests over all value pairs.

We reject the null hypothesis only if the p-value is lower than 10^{-4}. Holm's method is used to control the family-wise error rate when performing multiple hypothesis tests. This controls the probability of even a single false positive over all hypothesis tests. We always use the two-sided variant of a hypothesis test, since a bias can be either positive or negative.

Simply giving or plotting the probability of two dependent bytes is not ideal. After all, this probability includes the single-byte biases, while we only want to report the strength of the dependency between both bytes. To solve this, we report the absolute relative bias compared to the expected single-byte based probability. More precisely, say that by multiplying the two single-byte probabilities of a pair, we would expect it to occur with probability p. Given that this pair actually occurs with probability s, we then plot the value $|q|$ from the formula $s = p \cdot (1 + q)$. In a sense the relative bias indicates how much information is gained by not just considering the single-byte biases, but using the real byte-pair probability.

4.3.2 Generating datasets

In order to generate detailed statistics of keystream bytes, we created a distributed setup. We used roughly 80 standard desktop computers and three powerful servers as workers. The generation of the statistics is done in C. Python was used to manage the generated datasets and control all workers. On start-up each worker generates a cryptographically random AES key. Random 128-bit RC4 keys are derived from this key using AES in counter mode. Finally, we used R for all statistical analysis [136].

Our main results are based on two datasets, called `first16` and `consec512`. The `first16` dataset estimates $\Pr[Z_a = x \wedge Z_b = y]$ for $1 \leq a \leq 16$, $1 \leq b \leq 256$, and $0 \leq x, y < 256$ using 2^{44} keys. Its generation took roughly 9 CPU years. This allows detecting biases between the first 16 bytes and the other initial 256 bytes. The `consec512` dataset estimates $\Pr[Z_r = x \wedge Z_{r+1} = y]$ for $1 \leq r \leq 512$ and $0 \leq x, y < 256$ using 2^{45} keys, which took 16 CPU years to generate. It allows a detailed study of consecutive keystream bytes up to position 512.

We optimize the generation of both datasets. The first optimization is that

one run of a worker generates at most 2^{30} keystreams. This allows usage of 16-bit integers for all counters collecting the statistics, even in the presence of significant biases. Only when combining the results of workers are larger integers required. This lowers memory usage, reducing cache misses. To further reduce cache misses we generate several keystreams before updating the counters. In prior work, Paterson et al. used similar optimizations [124]. For the `first16` dataset we use an additional optimization. Here we first generate several keystreams, and then update the counters in a sorted manner based on the value of Z_a. This optimization caused the most significant speed-up for the `first16` dataset.

4.3.3 New short-term biases

By analysing the generated datasets we discovered many new short-term biases. We classify them into several sets.

Biases in (non-)consecutive bytes

By analysing the `consec512` dataset we discovered numerous biases between consecutive keystream bytes. Our first observation is that the Fluhrer-McGrew biases are also present in the initial keystream bytes. Exceptions occur at positions 1, 2 and 5, and are listed in Table 4.1 (note the extra conditions on the position r). This is surprising, as the Fluhrer-McGrew biases were generally not expected to be present in the initial keystream bytes [56]. However, these biases are present, albeit with different probabilities. Figure 4.4 shows the absolute relative bias of most Fluhrer-McGrew digraphs, compared to their expected single-byte based probability (recall Sect. 4.3.1). For all digraphs, the sign of the relative bias q is the same as its long-term variant as listed in Table 4.1. We observe that the relative biases converge to their long-term values, especially after position 257. The vertical lines around position 1 and 256 are caused by digraphs which do not hold (or hold more strongly) around these positions.

A second set of strong biases have the form:

$$\Pr[Z_{w16-1} = Z_{w16} = 256 - w16]\,, \tag{4.2}$$

with $1 \leq w \leq 7$. In Table 4.2 we list their probabilities. Since 16 equals our key length, these are likely key-length dependent biases.

Another set of biases has the form $\Pr[Z_r = Z_{r+1} = x]$. Depending on the value x, these biases are either negative or positive. Hence summing over all x and calculating $\Pr[Z_r = Z_{r+1}]$ would lose some statistical information.

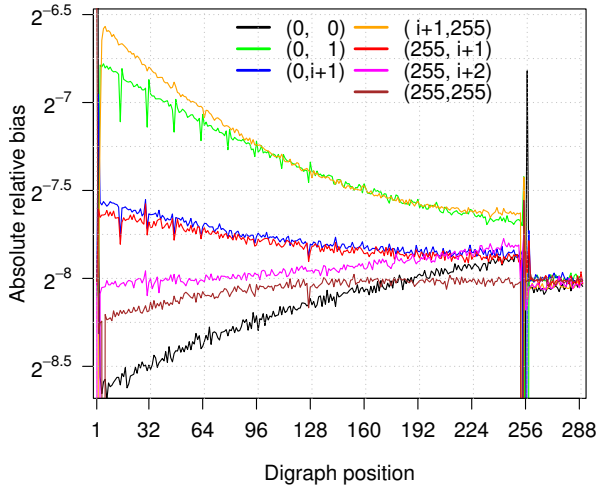

Figure 4.4: Absolute relative bias of several Fluhrer-McGrew digraphs in the initial keystream bytes, compared to their expected single-byte based probability.

In principle, these biases also include the Fluhrer-McGrew pairs $(0, 0)$ and $(255, 255)$. However, as the bias for both these pairs is much higher than for other values, we excluded them in our analysis. Our new bias, in the form of $\Pr[Z_r = Z_{r+1}]$, was detected up to position 512.

We also detected biases between non-consecutive bytes that do not fall in any obvious categories. An overview of these is given in Table 4.2. We remark that the biases induced by $Z_{16} = 240$ generally have a position, or value, that is a multiple of 16. This is an indication that these are likely key-length dependent biases.

Influence of Z_1 and Z_2

Arguably our most intriguing finding is the amount of information the first two keystream bytes leak. In particular, Z_1 and Z_2 influence all initial 256 keystream bytes. We detected the following six sets of biases:

$$
\begin{array}{lll}
1) & Z_1 = 257 - i \wedge Z_i = 0 & 4) \quad Z_1 = i - 1 \wedge Z_i = 1 \\
2) & Z_1 = 257 - i \wedge Z_i = i & 5) \quad Z_2 = 0 \wedge Z_i = 0 \\
3) & Z_1 = 257 - i \wedge Z_i = 257 - i & 6) \quad Z_2 = 0 \wedge Z_i = i
\end{array}
$$

Table 4.2: Biases between consecutive and non-consecutive bytes.

First byte	Second byte	Probability
Consecutive biases:		
$Z_{15} = 240$	$Z_{16} = 240$	$2^{-15.94786}(1 - 2^{-4.894})$
$Z_{31} = 224$	$Z_{32} = 224$	$2^{-15.96486}(1 - 2^{-5.427})$
$Z_{47} = 208$	$Z_{48} = 208$	$2^{-15.97595}(1 - 2^{-5.963})$
$Z_{63} = 192$	$Z_{64} = 192$	$2^{-15.98363}(1 - 2^{-6.469})$
$Z_{79} = 176$	$Z_{80} = 176$	$2^{-15.99020}(1 - 2^{-7.150})$
$Z_{95} = 160$	$Z_{96} = 160$	$2^{-15.99405}(1 - 2^{-7.740})$
$Z_{111} = 144$	$Z_{112} = 144$	$2^{-15.99668}(1 - 2^{-8.331})$
Non-consecutive biases:		
$Z_3 = 4$	$Z_5 = 4$	$2^{-16.00243}(1 + 2^{-7.912})$
$Z_3 = 131$	$Z_{131} = 3$	$2^{-15.99543}(1 + 2^{-8.700})$
$Z_3 = 131$	$Z_{131} = 131$	$2^{-15.99347}(1 - 2^{-9.511})$
$Z_4 = 5$	$Z_6 = 255$	$2^{-15.99918}(1 + 2^{-8.208})$
$Z_{14} = 0$	$Z_{16} = 14$	$2^{-15.99349}(1 + 2^{-9.941})$
$Z_{15} = 47$	$Z_{17} = 16$	$2^{-16.00191}(1 + 2^{-11.279})$
$Z_{15} = 112$	$Z_{32} = 224$	$2^{-15.96637}(1 - 2^{-10.904})$
$Z_{15} = 159$	$Z_{32} = 224$	$2^{-15.96574}(1 + 2^{-9.493})$
$Z_{16} = 240$	$Z_{31} = 63$	$2^{-15.95021}(1 + 2^{-8.996})$
$Z_{16} = 240$	$Z_{32} = 16$	$2^{-15.94976}(1 + 2^{-9.261})$
$Z_{16} = 240$	$Z_{33} = 16$	$2^{-15.94960}(1 + 2^{-10.516})$
$Z_{16} = 240$	$Z_{40} = 32$	$2^{-15.94976}(1 + 2^{-10.933})$
$Z_{16} = 240$	$Z_{48} = 16$	$2^{-15.94989}(1 + 2^{-10.832})$
$Z_{16} = 240$	$Z_{48} = 208$	$2^{-15.92619}(1 - 2^{-10.965})$
$Z_{16} = 240$	$Z_{64} = 192$	$2^{-15.93357}(1 - 2^{-11.229})$

Their absolute relative bias, compared to the single-byte biases, is shown in Fig. 4.5. The relative bias of pairs 5 and 6, i.e., those involving Z_2, are generally negative. Pairs involving Z_1 are generally positive, except pair 3, which always has a negative relative bias. We also detected dependencies between Z_1 and Z_2 other than the $\Pr[Z_1 = Z_2]$ bias of Paul and Preneel [127]. That is, the following pairs are strongly biased:

A) $Z_1 = 0 \wedge Z_2 = x$ C) $Z_1 = x \wedge Z_2 = 0$
B) $Z_1 = x \wedge Z_2 = 258 - x$ D) $Z_1 = x \wedge Z_2 = 1$

Bias A and C are negative for all $x \neq 0$, and both appear to be mainly caused by the strong positive bias $\Pr[Z_1 = Z_2 = 0]$ found by Isobe et al. Bias B and D

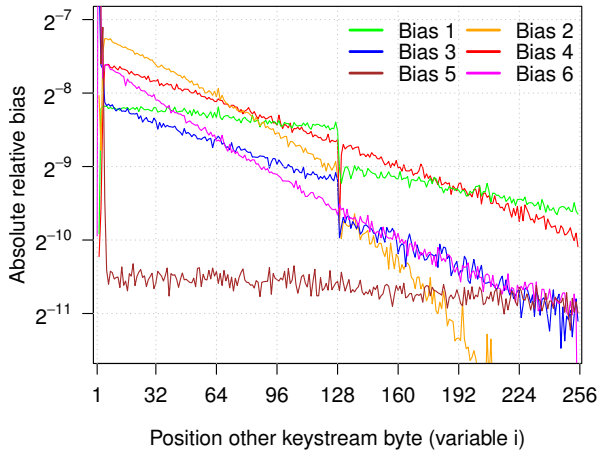

Figure 4.5: Biases induced by the first two bytes. The number of the biases correspond to those in Sect. 4.3.3.

are positive. We also discovered the following three biases:

$$\Pr[Z_1 = Z_3] = 2^{-8}(1 - 2^{-9.617}), \tag{4.3}$$

$$\Pr[Z_1 = Z_4] = 2^{-8}(1 + 2^{-8.590}), \tag{4.4}$$

$$\Pr[Z_2 = Z_4] = 2^{-8}(1 - 2^{-9.622}). \tag{4.5}$$

Note that all involve an equality either with Z_1 or Z_2.

Single-byte biases

We analyzed single-byte biases by aggregating the `consec512` dataset, and by generating additional statistics specifically for single-byte probabilities. The aggregation corresponds to calculating

$$\Pr[Z_r = k] = \sum_{y=0}^{255} \Pr[Z_r = k \wedge Z_{r+1} = y]. \tag{4.6}$$

We ended up with 2^{47} keys used to estimate single-byte probabilities. For all initial 513 bytes we could reject the hypothesis that they are uniformly distributed. In other words, all initial 513 bytes are biased. Figure 4.6 shows the probability distribution for some positions. Manual inspection of the

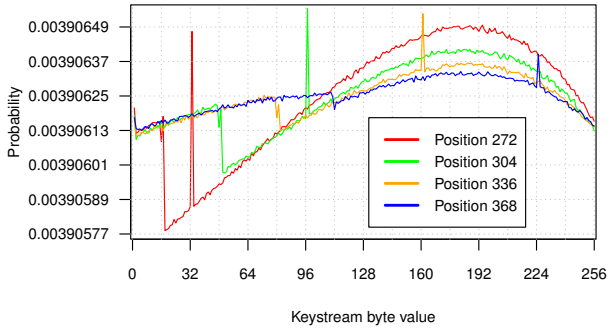

Figure 4.6: Single-byte biases beyond position 256.

distributions revealed a significant bias towards $Z_{256+k\cdot 16} = k \cdot 32$ for $1 \leq k \leq 7$. These are likely key-length dependent biases. Following [111] we conjecture there are single-byte biases even beyond these positions, albeit less strong.

4.3.4 New long-term biases

To search for new long-term biases we created a variant of the `first16` dataset. It estimates

$$\Pr[Z_{256w+a} = x \wedge Z_{256w+b} = y] \tag{4.7}$$

for $0 \leq a \leq 16$, $0 \leq b < 256$, $0 \leq x, y < 256$, and $w \geq 4$. It is generated using 2^{12} RC4 keys, where each key was used to generate 2^{40} keystream bytes. This took roughly 8 CPU years. The condition on w means we always dropped the initial 1023 keystream bytes. Using this dataset we can detect biases whose periodicity is a proper divisor of 256 (e.g., it detected all Fluhrer-McGrew biases). Our new short-term biases were not present in this dataset, indicating they indeed only occur in the initial keystream bytes, at least with the probabilities we listed. We did find the new long-term bias

$$\Pr[(Z_{w256}, Z_{w256+2}) = (128, 0)] = 2^{-16}(1 + 2^{-8}) \tag{4.8}$$

for $w \geq 1$. Surprisingly this was not discovered earlier, since a bias towards $(0, 0)$ at these positions was already known [154].

By aggregating our dataset we also detected several biases of the form $\Pr[Z_r = Z_{r+b}] = 2^{-8}(2 \pm 2^{-16})$. We call these type of biases distant-equality biases. Due to the small relative bias of 2^{-16}, these are difficult to reliably detect. We addressed this issue by generating a new dataset that specifically estimates $\Pr[Z_{256w+a} = Z_{256w+b}]$ for $0 \leq a \leq 16$, $0 \leq b < 256$, $0 \leq x, y < 256$,

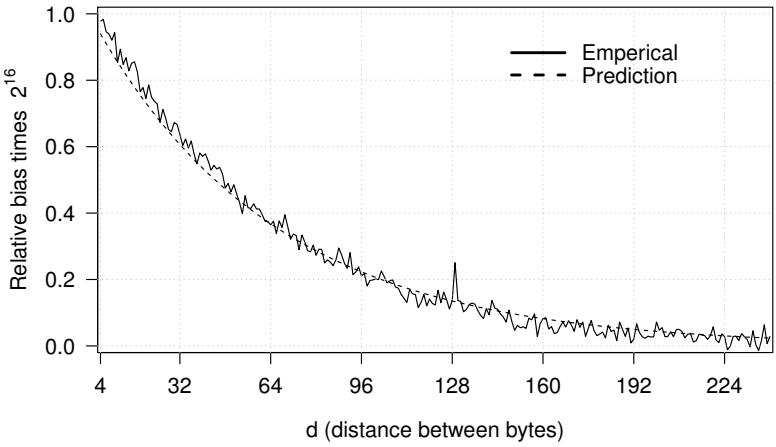

Figure 4.7: Empirical relative bias times 2^{16} of the long-term distant-equality bias, in function of the distance d, for $d \geq 4$. In other words, this is the value of the function $\delta(d)$ in equation (4.11).

and $w \geq 4$. It is generated using roughly 2^{15} RC4 keys, where each key was used to generate 2^{40} keystream bytes. In total this dataset took 17 CPU years to generate. By analysing it we found that these biases are independent of their position in the keystream, i.e., they do not depend on the public counter i. The discovered set of biases can be summarized as follows:

$$\Pr[Z_r = Z_{r+1}] \approx 2^{-8}(1 + 2^{-16}), \tag{4.9}$$

$$\Pr[Z_r = Z_{r+2}] \approx 2^{-8}(1 - 2^{-16}), \tag{4.10}$$

$$\Pr[Z_r = Z_{r+d} \mid d \geq 4] \approx 2^{-8}(1 + \delta(d) \cdot 2^{-16}). \tag{4.11}$$

Here $\delta(d)$ is a decreasing function, whose value is plotted in Fig. 4.7. The dashed line shows the value of the function $f(d) = \mathrm{e}^{-4d/256}$. This formula was inspired by Lemma 1 of Mantin in [99]. It states that, with a probably of $\mathrm{e}^{-rd/256}$, r locations in the RC4 state will have the same value d rounds later. Unfortunately, we do not yet have an exact explanation why our formula closely matches our empirical observations, and leave a proper explanation as future work. Interestingly, no bias was detected for the event $Z_r = Z_{r+3}$, i.e., keystream bytes that are three positions apart are not noticeably correlated.

At this point it is worth noting that Paul and Preneel theoretically predicted the following bias [127, §2.4]:

$$\Pr[Z_{256w+1} = Z_{256w+2}] = 2^{-8}(1 - 2^{-16}). \tag{4.12}$$

They empirically confirmed this for $w = 1$. However, our results show that this prediction no longer holds for larger values of w.

Note that the positive bias for the event $Z_r = Z_{r+1}$ cannot be explained by the Fluhrer-McGrew biases. In particular, for most values of i, the positive bias for digraph $(0, 0)$ would be cancelled out by the negative bias for digraph $(255, 255)$. This would imply the bias for $Z_r = Z_{r+1}$ would not be present for most values of i, which contradicts our empirical observations.

After replicating the work of Fluhrer and McGrew [56], and counting the number of internal RC4 states consistent with the event $Z_r = Z_{r+1}$ for scaled down versions of RC4, we did not detect a bias for this event. We took into account that RC4 will never enter a Finney state while doing this [54]. The only assumption made by the method of Fluhrer and McGrew, is that every state which RC4 can enter is equally likely [56]. Since we did not rediscover the bias towards $Z_r = Z_{r+1}$, it means this assumption does not hold. In other words, the bias towards $Z_r = Z_{r+1}$ may be caused by currently unknown (long-term) biases in the state of RC4 itself. We consider it an interesting future research direction to further investigate this conjecture.

4.4 Plaintext Recovery

We will design plaintext recovery techniques for usage in two areas: decrypting TKIP packets and HTTPS cookies. In other scenarios, variants of our methods can be used.

4.4.1 Calculating likelihood estimates

Our goal is to convert a sequence of ciphertexts $\mathcal{C}$ into predictions about the plaintext. This is done by exploiting biases in the keystream distributions $p_k = \Pr[Z_r = k]$. These can be obtained by following the steps in Sect. 4.3.2. All biases in p_k are used to calculate the likelihood that a plaintext byte equals a certain value μ. To accomplish this, we rely on the likelihood calculations of AlFardan et al. [5]. Their idea is to calculate, for each plaintext value μ, the (induced) keystream distributions required to witness the captured ciphertexts. The closer this matches the real keystream distributions p_k, the more likely we have the correct plaintext byte. Assuming a fixed position r for simplicity, the induced keystream distributions are defined by the vector $N^\mu = (N_0^\mu, \ldots, N_{255}^\mu)$. Each N_k^μ represents the number of times the keystream byte was equal to k,

assuming the plaintext byte was μ:

$$N_k^\mu = |\{C \in \mathcal{C} \mid C = k \oplus \mu\}| \,. \tag{4.13}$$

Note that the vectors N^μ and $N^{\mu'}$ are permutations of each other. Based on the real keystream probabilities p_k we calculate the likelihood that this induced distribution would occur in practice. This is modelled using a multinomial distribution with the number of trails equal to $|\mathcal{C}|$, and the categories being the 256 possible keystream byte values. Since we want the probability of this *sequence* of keystream bytes we get [124]:

$$\Pr[\mathcal{C} \mid P = \mu] = \prod_{k \in \{0,\ldots,255\}} (p_k)^{N_k^\mu} \,. \tag{4.14}$$

Using Bayes' theorem we can convert this into the likelihood λ_μ that the plaintext byte is μ:

$$\lambda_\mu = \Pr[P = \mu \mid \mathcal{C}] \sim \Pr[\mathcal{C} \mid P = \mu] \,. \tag{4.15}$$

For our purposes we can treat this as an equality [5]. The most likely plaintext byte μ is the one that maximises λ_μ. This was extended to a pair of dependent keystream bytes in the obvious way:

$$\lambda_{\mu_1,\mu_2} = \prod_{k_1,k_2 \in \{0,\ldots,255\}} (p_{k_1,k_2})^{N_{k_1,k_2}^{\mu_1,\mu_2}} \,. \tag{4.16}$$

We found this formula can be optimized if most keystream byte values k_1 and k_2 are independent and uniform. More precisely, let us assume that all keystream value pairs in the set $\mathcal{I}$ are independent and uniform:

$$\forall (k_1,k_2) \in \mathcal{I} \colon p_{k_1,k_2} = p_{k_1} \cdot p_{k_2} = u \,, \tag{4.17}$$

where u represents the probability of an unbiased double-byte keystream value. Then we rewrite equation (4.16) to:

$$\lambda_{\mu_1,\mu_2} = (u)^{M^{\mu_1,\mu_2}} \cdot \prod_{k_1,k_2 \in \mathcal{I}^c} (p_{k_1,k_2})^{N_{k_1,k_2}^{\mu_1,\mu_2}} \,, \tag{4.18}$$

where

$$M^{\mu_1,\mu_2} = \sum_{k_1,k_2 \in \mathcal{I}} N_{k_1,k_2}^{\mu_1,\mu_2} = |\mathcal{C}| - \sum_{k_1,k_2 \in \mathcal{I}^c} N_{k_1,k_2}^{\mu_1,\mu_2} \tag{4.19}$$

and with $\mathcal{I}^c$ the set of dependent keystream values. If the set $\mathcal{I}^c$ is small, this results in a lower time-complexity. For example, when applied to the long-term keystream setting over Fluhrer-McGrew biases, roughly 2^{19} operations are required to calculate all likelihood estimates, instead of 2^{32}. A similar (though less drastic) optimization can also be made when single-byte biases are present.

4.4.2 Likelihoods from Mantin's bias

We now show how to compute a double-byte plaintext likelihood using Mantin's $AB\mathcal{S}AB$ bias. More formally, we want to compute the likelihood λ_{μ_1,μ_2} that the plaintext bytes at fixed positions r and $r+1$ are μ_1 and μ_2, respectively. To accomplish this we abuse surrounding known plaintext. Our main idea is to first calculate the likelihood of the *differential* between the known and unknown plaintext. We define the differential $\widehat{Z}_r^g$ as:

$$\widehat{Z}_r^g = (Z_r \oplus Z_{r+2+g}, Z_{r+1} \oplus Z_{r+3+g})\,. \tag{4.20}$$

Similarly we use $\widehat{C}_r^g$ and $\widehat{P}_r^g$ to denote the differential over ciphertext and plaintext bytes, respectively. The $AB\mathcal{S}AB$ bias of (4.1) can then be written as:

$$\Pr[\widehat{Z}_r^g = (0,0)] = 2^{-16}(1 + 2^{-8}\mathrm{e}^{\frac{-4-8g}{256}}) = \alpha(g)\,. \tag{4.21}$$

When XORing both sides of $\widehat{Z}_r^g = (0,0)$ with $\widehat{P}_r^g$ we get

$$\Pr[\widehat{C}_r^g = \widehat{P}_r^g] = \alpha(g)\,. \tag{4.22}$$

Hence Mantin's bias implies that the ciphertext differential is biased towards the plaintext differential. We use this to calculate the likelihood $\lambda_{\hat{\mu}}$ of a plaintext differential $\hat{\mu}$. For ease of notation we assume a fixed position r and a fixed $AB\mathcal{S}AB$ gap of g. Let $\widehat{\mathcal{C}}$ be the sequence of captured ciphertext differentials, and μ_1' and μ_2' the known plaintext bytes at positions $r+2+g$ and $r+3+g$, respectively. Similar to our previous likelihood estimates, we calculate the probability of witnessing the ciphertext differentials $\widehat{\mathcal{C}}$ assuming the plaintext differential is $\hat{\mu}$:

$$\Pr[\widehat{\mathcal{C}} \mid \widehat{P} = \hat{\mu}] = \prod_{\hat{k} \in \{0,\ldots,255\}^2} \Pr[\widehat{Z} = \hat{k}]^{N_{\hat{k}}^{\hat{\mu}}}\,, \tag{4.23}$$

where

$$N_{\hat{k}}^{\hat{\mu}} = \left|\left\{\widehat{C} \in \widehat{\mathcal{C}} \mid \widehat{C} = \hat{k} \oplus \hat{\mu}\right\}\right|\,. \tag{4.24}$$

Using this notation we see that this is indeed identical to an ordinary likelihood estimation. Using Bayes' theorem we get $\lambda_{\hat{\mu}} = \Pr[\widehat{P} = \hat{\mu} \mid \widehat{\mathcal{C}}] \sim \Pr[\widehat{\mathcal{C}} \mid \widehat{P} = \hat{\mu}]$. Since only one differential pair is biased, we can apply and simplify equation (4.18):

$$\lambda_{\hat{\mu}} = (1 - \alpha(g))^{|\mathcal{C}|-|\hat{u}|} \cdot \alpha(g)^{|\hat{\mu}|}\,, \tag{4.25}$$

where we slightly abuse notation by defining $|\hat{\mu}|$ as

$$|\hat{\mu}| = \left|\left\{\widehat{C} \in \widehat{\mathcal{C}} \mid \widehat{C} = \hat{\mu}\right\}\right| \tag{4.26}$$

Finally we apply our knowledge of the known plaintext bytes to get our desired likelihood estimate:

$$\lambda_{\mu_1,\mu_2} = \lambda_{\hat{\mu}\oplus(\mu'_1,\mu'_2)} \,. \tag{4.27}$$

To estimate at which gap size the $AB\mathcal{S}AB$ bias is still detectable, we generated 2^{48} blocks of 512 keystream bytes. Based on this we empirically confirmed Mantin's $AB\mathcal{S}AB$ bias up to gap sizes of at least 135 bytes. The theoretical estimate in equation (4.1) slightly underestimates the true empirical bias. In our attacks we use a maximum gap size of 128.

4.4.3 Combining likelihood estimates

Our goal is to combine multiple types of biases in a likelihood calculation. Unfortunately, if the biases cover overlapping positions, it quickly becomes infeasible to perform a single likelihood estimation over all bytes. In the worst case, the calculation cannot be optimized by relying on independent biases. Hence, a likelihood estimate over n keystream positions would have a time complexity of $\mathcal{O}(2^{2\cdot 8\cdot n})$. To overcome this problem, we perform and combine multiple separate likelihood estimates.

We will combine multiple types of biases by multiplying their individual likelihood estimates. For example, let λ'_{μ_1,μ_2} be the likelihood of plaintext bytes μ_1 and μ_2 based on the Fluhrer-McGrew biases. Similarly, let λ'_{g,μ_1,μ_2} be likelihoods derived from $AB\mathcal{S}AB$ biases of gap g. Then their combination is straightforward:

$$\lambda_{\mu_1,\mu_2} = \lambda'_{\mu_1,\mu_2} \cdot \prod_g \lambda'_{g,\mu_1,\mu_2} \,. \tag{4.28}$$

While this method may not be optimal when combining likelihoods of dependent bytes, it does appear to be a general and powerful method. An open problem is determining which biases can be combined under a single likelihood calculation, while keeping computational requirements acceptable. Likelihoods based on other biases, e.g., Sen Gupta's and our new long-term biases, can be added as another factor (though some care is needed so positions properly overlap).

To verify the effectiveness of this approach, we performed simulations where we attempt to decrypt two bytes using one double-byte likelihood estimate. First this is done using only the Fluhrer-McGrew biases, and using only one $AB\mathcal{S}AB$ bias. Then we combine $2 \cdot 129$ $AB\mathcal{S}AB$ biases and the Fluhrer-McGrew biases, where we use the method from Sect. 4.4.2 to derive likelihoods from $AB\mathcal{S}AB$ biases. We assume the unknown bytes are surrounded at both sides by known plaintext, and use a maximum $AB\mathcal{S}AB$ gap of 128 bytes. Additionally, we assume these two bytes can take on any values, i.e., we assume an unrestricted

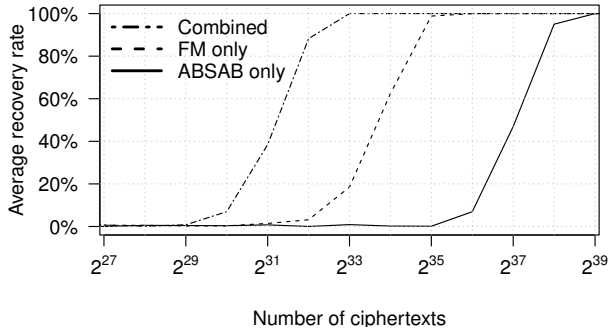

Figure 4.8: Average success rate of decrypting two bytes using: (1) one $ABSAB$ bias; (2) Fluhrer-McGrew (FM) biases; and (3) combination of FM biases with 258 $ABSAB$ biases. Results based on 2048 simulations each.

plaintext space. Figure 4.8 shows the results of this experiment. Notice that a single $ABSAB$ bias is weaker than using all Fluhrer-McGrew biases at a specific position. However, combining several $ABSAB$ biases clearly results in a major improvement. We conclude that our approach to combine biases significantly reduces the required number of ciphertexts.

4.4.4 List of plaintext candidates

In practice it is useful to have a list of plaintext candidates in decreasing likelihood. For example, by traversing this list we could attempt to brute-force keys, passwords, cookies, etc. (see Sect. 4.6). In other situations the plaintext may have a rigid structure allowing the removal of candidates (see Sect. 4.5). We will generate a list of plaintext candidates in decreasing likelihood, when given either single-byte or double-byte likelihood estimates.

First we show how to construct a candidate list when given single-byte plaintext likelihoods. While it is trivial to generate the two most likely candidates, beyond this point the computation becomes more tedious. Our solution is to incrementally compute the N most likely candidates based on their length. That is, we first compute the N most likely candidates of length 1, then of length 2, and so on. Algorithm 1 gives a high-level implementation of this idea. Variable $P_r[i]$ denotes the i-th most likely plaintext of length r, having a likelihood of $E_r[i]$. The two `min` operations are needed because in the initial loops we are not yet be able to generate N candidates, i.e., there only exist 256^r plaintexts of length r. Picking the μ' which maximizes $pr(\mu')$ can be done efficiently using a

Algorithm 1: Generate plaintext candidates in decreasing likelihood using single-byte estimates.

Input: L : Length of the unknown plaintext
$\lambda_{1 \le r \le L,\ 0 \le \mu \le 255}$: single-byte likelihoods
N: Number of candidates to generate
Returns: List of candidates in decreasing likelihood

begin
  $P_0[1] \leftarrow \epsilon$
  $E_0[1] \leftarrow 0$
  for $r = 1$ **to** L **do**
    for $\mu = 0$ **to** 255 **do**
      $pos(\mu) \leftarrow 1$
      $pr(\mu) \leftarrow E_{r-1}[1] + \log(\lambda_{r,\mu})$
    for $i = 1$ **to** $\min(N, 256^r)$ **do**
      $\mu \leftarrow \mu'$ which maximizes $pr(\mu')$
      $P_r[i] \leftarrow P_{r-1}[pos(\mu)] \parallel \mu$
      $E_r[i] \leftarrow E_{r-1}[pos(\mu)] + \log(\lambda_{r,\mu})$
      $pos(\mu) \leftarrow pos(\mu) + 1$
      $pr(\mu) \leftarrow E_{r-1}[pos(\mu)] + \log(\lambda_{r,\mu})$
      if $pos(\mu) > \min(N, 256^{r-1})$ **then**
        $pr(\mu) \leftarrow -\infty$
  return P_N

priority queue. In practice, only the latest two versions of lists E and P need to be stored. To better maintain numeric stability, and to make the computation more efficient, we perform calculations using the logarithm of the likelihoods. We implemented Algorithm 1 and report on its performance in Sect. 4.5, where we use it to attack a wireless network protected by WPA-TKIP.

To generate a list of candidates from double-byte likelihoods, we first show how to model the likelihoods as a hidden Markov model (HMM). This allows us to present a more intuitive version of our algorithm, and refer to the extensive research in this area if more efficient implementations are needed. The algorithm we present can be seen as a combination of the classical Viterbi algorithm, and Algorithm 1. Even though it is not the most optimal one, it still proved sufficient to significantly improve plaintext recovery (see Sect. 4.6). For an introduction to HMMs we refer the reader to [137]. Essentially an HMM models a system where the internal states are not observable, and after each state transition, output is (probabilistically) produced dependent on its new state.

We model the plaintext likelihood estimates as a first-order time-inhomogeneous HMM. The state space S of the HMM is defined by the set of possible plaintext values $\{0, \ldots, 255\}$. The byte positions are modelled using the time-dependent (i.e., inhomogeneous) state transition probabilities. Intuitively, the "current time" in the HMM corresponds to the current plaintext position. This means the transition probabilities for moving from one state to another, which normally depend on the current time, will now depend on the position of the byte. More formally:

$$\Pr[S_{t+1} = \mu_2 \mid S_t = \mu_1] \sim \lambda_{t,\mu_1,\mu_2}\,, \tag{4.29}$$

where t represents the time. For our purposes we can treat this as an equality. In an HMM it is assumed that its current state is not observable. This corresponds to the fact that we do not know the value of any plaintext bytes. In an HMM there is also some form of output which depends on the current state. In our setting a particular plaintext value leaks no observable (side-channel) information. This is modelled by always letting every state produce the same null output with probability one.

Using the above HMM model, finding the most likely plaintext reduces to finding the most likely state sequence. This is solved using the well-known Viterbi algorithm. Indeed, the algorithm presented by AlFardan et al. closely resembles the Viterbi algorithm [5]. Similarly, finding the N most likely plaintexts is the same as finding the N most likely state sequences. Hence any N-best variant of the Viterbi algorithm (also called list Viterbi algorithm) can be used, examples being [118, 146, 157, 161]. The simplest form of such an algorithm keeps track of the N best candidates ending in a particular value μ, and is shown in Algorithm 2. Similar to [5, 124] we assume the first byte m_1 and last byte m_L of the plaintext are known. During the last round of the outer for-loop, the loop over μ_2 has to be executed only for the value m_L. In Sect. 4.6 we use this algorithm to generate a list of cookies.

Algorithm 2 uses considerably more memory than Algorithm 1. This is because it has to store the N most likely candidates for each possible ending value μ. We remind the reader that our goal is not to present the most optimal algorithm. Instead, by showing how to model the problem as an HMM, we can rely on related work in this area for more efficient algorithms [118, 146, 157, 161]. Since an HMM can be modelled as a graph, all k-shortest path algorithms are also applicable [49]. Finally, we remark that even our simple variant sufficed to significantly improve plaintext recovery rates (see Sect. 4.6).

Algorithm 2: Generate plaintext candidates in decreasing likelihood using double-byte estimates.

Input: L : Length of the unknown plaintext plus two
m_1 and m_L: known first and last byte
$\lambda_{1\leq r<L,\ 0\leq\mu_1,\mu_2\leq 255}$: double-byte likelihoods
N: Number of candidates to generate
Returns: List of candidates in decreasing likelihood

begin
  for $\mu_2 = 0$ **to** 255 **do**
    $E_2[\mu_2, 1] \leftarrow \log(\lambda_{1,m_1,\mu_2})$
    $P_2[\mu_2, 1] \leftarrow m_1 \parallel \mu_2$
  for $r = 3$ **to** L **do**
    for $\mu_2 = 0$ **to** 255 **do**
      for $\mu_1 = 0$ **to** 255 **do**
        $pos(\mu_1) \leftarrow 1$
        $pr(\mu_1) \leftarrow E_{r-1}[\mu_1, 1] + \log(\lambda_{r,\mu_1,\mu_2})$
      for $i = 1$ **to** $\min(N, 256^{r-1})$ **do**
        $\mu_1 \leftarrow \mu$ which maximizes $pr(\mu)$
        $P_r[\mu_2, i] \leftarrow P_{r-1}[\mu_1, pos(\mu_1)] \parallel \mu_2$
        $E_r[\mu_2, i] \leftarrow E_{r-1}[\mu_1, pos(\mu_1)] + \log(\lambda_{r,\mu_1,\mu_2})$
        $pos(\mu_1) \leftarrow pos(\mu_1) + 1$
        $pr(\mu_1) \leftarrow E_{r-1}[\mu_1, pos(\mu_1)] + \log(\lambda_{r,\mu_1,\mu_2})$
        if $pos(\mu_1) > \min(N, 256^{r-2})$ **then**
          $pr(\mu_1) \leftarrow -\infty$
  return $P_N[m_L, :]$

4.5 Attacking WPA-TKIP

We use our plaintext recovery techniques to decrypt a full packet. From this decrypted packet the MIC key can be derived, allowing an attacker to inject and decrypt packets. The attack takes only an hour to execute in practice.

4.5.1 Calculating plaintext likelihoods

We rely on the attack of Paterson et al. to compute plaintext likelihood estimates [124, 125]. They noticed that the first three bytes of the per-packet RC4 key are public. As explained in Sect. 4.2.2, the first three bytes are fully

determined by the TKIP Sequence Counter (TSC). It was observed that this dependency causes strong TSC-dependent biases in the keystream [67, 124, 125], which can be used to improve the plaintext likelihood estimates. For each TSC value they calculated plaintext likelihoods based on empirical, per-TSC, keystream distributions. The resulting 256^2 likelihoods, for one plaintext byte, are combined by multiplying them over all TSC pairs. In a sense this is similar to combining multiple types of biases as done in Sect. 4.4.3, though here the different types of biases are known to be independent. We use the single-byte variant of the attack [124, §4.1] to obtain likelihoods $\lambda_{r,\mu}$ for every unknown byte at a given position r.

The downside of this attack is that it requires detailed per-TSC keystream statistics. Paterson at al. generated statistics for the first 512 bytes, which took 30 CPU years [124]. However, in our attack we only need these statistics for the first few keystream bytes. We used 2^{32} keys per TSC value to estimate the keystream distribution for the first 128 bytes. Using our distributed setup the generation of these statistics took 10 CPU years.

With our per-TSC keystream distributions we obtained similar results to those of Paterson et al. [124, 125]. By running simulations we confirmed that the odd byte positions [124, 125] can be recovered with a higher probability than others. Similarly, the bytes at positions 49-51 and 63-67 are generally recovered with higher probability as well. Both observations will be used to optimize the attack in practice.

4.5.2 Injecting identical packets

We show how to fulfil the first requirement of a successful attack: the generation of identical packets. If the IP of the victim is known, and incoming connections towards it are not blocked, we can simply send identical packets to the victim. Otherwise we induce the victim into opening a TCP connection to an attacker-controlled server. This connection is then used to transmit identical packets to the victim. A straightforward way to accomplish this is by social engineering the victim into visiting a website hosted by the attacker. The browser will open a TCP connection with the server in order to load the website. However, we can also employ more sophisticated methods that have a broader target range. One such method is abusing the inclusion of (insecure) third-party resources on popular websites [117]. For example, an attacker can register a mistyped domain, accidentally used in a resource address (e.g., an image URL) on a popular website. Whenever the victim visits this website and loads the resource, a TCP connection is made to the server of the attacker. In [117] these types of vulnerabilities were found to be present on several popular websites.

Additionally, any type of web vulnerability that can be abused to make a victim execute JavaScript can be utilised. In this sense, our requirements are more relaxed than those of the recent attacks on SSL and TLS, which *require* the ability to run JavaScript code in the victim's browser [5, 6, 46]. Another method is to hijack an existing TCP connection of the victim, which under certain conditions is possible without a man-in-the-middle position [71]. We conclude that, while there is no universal method to accomplish this, we can assume that we have (or somehow can control) a TCP connection with the victim. Using this connection we inject identical packets by repeatedly retransmitting the same TCP packet. Since retransmissions are valid TCP behaviour, this will work even if the victim is behind a firewall.

We now determine the optimal structure of the injected packet. A naive approach would be to use the shortest possible packet, meaning no TCP payload is included. Since the total size of the LLC/SNAP, IP, and TCP header is 48 bytes, the MIC and ICV would be located at position 49 up to and including 60 (see Fig. 4.2). At these locations 7 bytes are strongly biased. In contrast, if we use a TCP payload of 7 bytes, the MIC and ICV are located at position 56 up to and including 67. In this range 8 bytes are strongly biased, resulting in better plaintext likelihood estimates. Through simulations we confirmed that using a 7 byte payload increases the probability of successfully decrypting the MIC and ICV. In practice, adding 7 bytes of payload also makes the length of our injected packet unique. As a result we can easily identify and capture such packets. Given both these advantages, we use a TCP data packet containing 7 bytes of payload.

4.5.3 Decrypting a complete packet

Our goal is to decrypt the injected TCP packet, including its MIC and ICV fields. Note that all these TCP packets will be encrypted with a different RC4 key. For now we assume all fields in the IP and TCP packet are known, and we will later show why we can safely make this assumption. Hence, only the 8-byte MIC and 4-byte ICV of the packet remain unknown. We use the per-TSC keystream statistics to compute single-byte plaintext likelihoods for all 12 bytes. However, this alone would give a very low success probability of simultaneously (correctly) decrypting all bytes. We solve this by realising that the TKIP ICV is a simple CRC checksum which we can easily verify ourselves. Hence we can detect bad candidates by inspecting their CRC checksum. We now generate a plaintext candidate list, and traverse it until we find a packet having a correct CRC. This drastically improves the probability of simultaneously decrypting all bytes. From the decrypted packet we can derive the TKIP MIC key [164], which can then be used to inject and decrypt arbitrary packets (see Chapter 2).

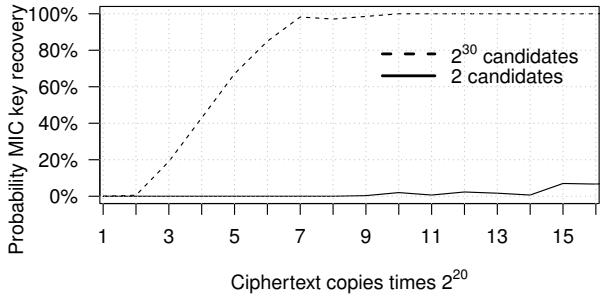

Figure 4.9: Success rate of obtaining the TKIP MIC key using nearly 2^{30} candidates, and using only the two best candidates. Results are based on 256 simulations each.

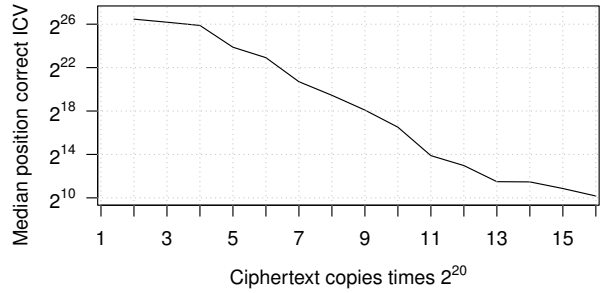

Figure 4.10: Median position of a candidate with a correct ICV with nearly 2^{30} candidates. Results are based on 256 simulations each.

Figure 4.9 shows the success rate of finding a packet with a good ICV and deriving the correct MIC key. For comparison, it also includes the success rates had we only used the two most likely candidates. Figure 4.10 shows the median position of the first candidate with a correct ICV. We plot the median instead of average to lower influence of outliers, i.e., at times the correct candidate was unexpectedly far (or early) in the candidate list.

The question that remains how to determine the contents of the unknown fields in the IP and TCP headers of the packet. More precisely, the unknown fields are the internal IP and port of the client, and the IP time-to-live (TTL) field. One observation makes this clear: both the IP and TCP header contain checksums. Therefore, we can apply exactly the same technique (i.e., candidate generation and pruning) to derive the values of these fields with high success rates. This can be done independently of each other, and independently of decrypting the MIC and ICV.

Another method to obtain the internal IP is to rely on HTML5 features. If the initial TCP connection is created by a browser, we can first send JavaScript code to obtain the internal IP of the victim using WebRTC [147]. We also noticed that our NAT gateway generally did not modify the source port used by the victim. Consequently we can simply read this value at the server. The TTL field can also be determined without relying on the IP checksum. Using a `traceroute` command we count the number of hops between the server and the client, allowing us to derive the TTL value at the victim.

4.5.4 Empirical evaluation

To test the plaintext recovery phase of our attack we created a tool that parses a raw `pcap` file containing the captured Wi-Fi packets. It searches for the injected packets, extracts the ciphertext statistics, calculates plaintext likelihoods, and searches for a candidate with a correct ICV. From this candidate, i.e., decrypted injected packet, we derive the MIC key.

For the ciphertext generation phase we used an OpenVZ VPS as malicious server. The incoming TCP connection from the victim is handled using a custom tool written in Scapy. It relies on a patched version of Tcpreplay to rapidly inject the identical TCP packets. The victim machine is a Latitude E6500 and is connected to an Asus RT-N10 router running Tomato 1.28. The victim opens a TCP connection to the malicious server by visiting a website hosted on it. For the attacker we used a Compaq 8510p with an AWUS036nha to capture the wireless traffic. Under this setup we were able to generate roughly 2500 packets per second. This number was reached even when the victim was actively browsing YouTube videos. Thanks to the 7-byte payload, we uniquely detected the injected packet in all experiments without any false positives.

We ran several tests where we generated and captured traffic for (slightly more) than one hour. This amounted to, on average, capturing $9.5 \cdot 2^{20}$ different encryptions of the packet being injected. Retransmissions were filtered based on the TSC of the packet. In nearly all cases we successfully decrypted the packet and derived the MIC key. Recall from Sect. 4.2.2 that this MIC key is valid as long as the victim does not renew its PTK, and that it can be used to inject and decrypt packets from the AP to the victim. For one capture our tool found a packet with a correct ICV, but this candidate did not correspond to the actual plaintext. While our current evaluation is limited in the number of captures performed, it shows the attack is practically feasible, with overall success probabilities appearing to agree with the simulated results of Fig. 4.9.

Listing 4.3: Manipulated HTTP request, with known plaintext surrounding the cookie at both sides.

```
1 GET / HTTP/1.1
2 Host: site.com
3 User-Agent: Mozilla/5.0 (X11; Linux i686; rv:32.0) Gecko/20100101 Firefox/32.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Cookie: auth=XXXXXXXXXXXXXXXX; injected1=known1; injected2=knownplaintext2; ...
```

4.6 Decrypting HTTPS Cookies

We inject known data around a cookie, enabling use of the $AB\mathcal{S}AB$ biases. We then show that a HTTPS cookie can be brute-forced using only 75 hours of ciphertext.

4.6.1 Injecting known plaintext

We want to be able to predict the position of the targeted cookie in the encrypted HTTP requests, and surround it with known plaintext. To fix ideas, we do this for the secure `auth` cookie sent to `https://site.com`. Similar to previous attacks on SSL and TLS, we assume the attacker is able to execute JavaScript code in the victim's browser [5, 6, 46]. In our case, this means an active man-in-the-middle (MitM) position is used, where plaintext HTTP channels can be manipulated. Our first realisation is that an attacker can predict the length and content of HTTP headers preceding the `Cookie` field. By monitoring plaintext HTTP requests, these headers can be sniffed. If the targeted `auth` cookie is the first value in the `Cookie` header, this implies we know its position in the HTTP request. Hence, our goal is to have a layout as shown in Listing 4.3. Here the targeted cookie is the first value in the `Cookie` header, preceded by known headers, and followed by attacker injected cookies.

To obtain the layout in Listing 4.3 we use our MitM position to redirect the victim to `http://site.com`, i.e., to the target website over an insecure HTTP channel. If the target website uses HTTP Strict Transport Security (HSTS), but does not use the `includeSubDomains` attribute, this is still possible by redirecting the victim to a (fake) subdomain [24]. Since few websites use HSTS, and even fewer use it properly [171], this redirection will likely succeed. Against old browsers HSTS can even be bypassed completely [20, 24, 160]. Since secure cookies guarantee only confidentiality but not integrity, the insecure HTTP

channel can be used to overwrite, remove, or inject secure cookies [15, p. 4.1.2.5]. This allows us to remove all cookies except the `auth` cookie, pushing it to the front of the list. After this we can inject cookies that will be included after the `auth` cookie. An example of a HTTP(S) request manipulated in this manner is shown in Listing 4.3. Here the secure `auth` cookie is surrounded by known plaintext at both sides. This allows us to use Mantin's $AB\mathcal{S}AB$ bias when calculating plaintext likelihoods.

4.6.2 Brute-forcing the cookie

In contrast to passwords, many websites do not protect against brute-forcing cookies. One reason for this is that the password of an average user has a much lower entropy than a random cookie. Hence it makes sense to brute-force a password, but not a cookie: the chance of successfully brute-forcing a (properly generated) cookie is close to zero. However, if RC4 can be used to connect to the web server, our candidate generation algorithm voids this assumption. We can traverse the plaintext candidate list in an attempt to brute-force the cookie.

Since we are targeting a cookie, we can exclude certain plaintext values. As RFC 6265 states, a cookie value can consists of at most 90 unique characters [15, §4.1.1]. A similar though less general observation was already made by AlFardan et al. [5]. Our observation allows us to give a tighter bound on the required number of ciphertexts to decrypt a cookie, even in the general case. In practice, executing the attack with a reduced character set is done by modifying Algorithm 2 so the for-loops over μ_1 and μ_2 only loop over allowed characters.

Figure 4.11 shows the success rate of brute-forcing a 16-character cookie using at most 2^{23} attempts. For comparison, we also include the probability of decrypting the cookie if only the most likely plaintext was used. This also allows for an easier comparison with the work for AlFardan et al. [5]. Note that they only use the Fluhrer-McGrew biases, whereas we combine serveral $AB\mathcal{S}AB$ biases together with the Fluhrer-McGrew biases. We conclude that our brute-force approach, as well as the inclusion of the $AB\mathcal{S}AB$ biases, significantly improves success rates. Even when using only 2^{23} brute-force attempts, success rates of more than 94% are obtained once $9 \cdot 2^{27}$ encryptions of the cookie have been captured. We conjecture that generating more candidates will further increase success rates.

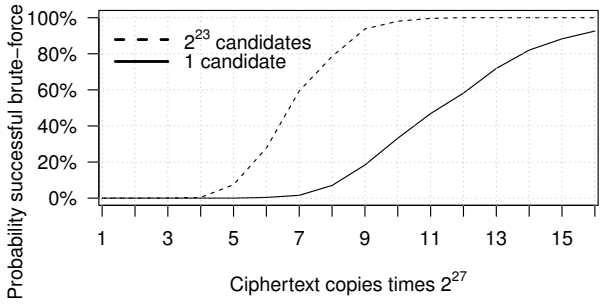

Figure 4.11: Success rate of brute-forcing a 16-character cookie using roughly 2^{23} candidates, and only the most likely candidate, dependent on the number of collected ciphertexts. Results based on 256 simulations each.

4.6.3 Empirical evaluation

The main requirement of our attack is being able to collect sufficiently many encryptions of the cookie, i.e., having many ciphertexts. We fulfil this requirement by forcing the victim to generate a large number of HTTPS requests. As in previous attacks on TLS [5, 6, 46], we accomplish this by assuming the attacker is able to execute JavaScript in the browser of the victim. For example, when performing a man-in-the-middle attack, we can inject JavaScript into any plaintext HTTP connection. We then use `XMLHttpRequest` objects to issue Cross-Origin Requests to the targeted website. The browser will automatically add the secure cookie to these (encrypted) requests. Due to the same-origin policy we cannot read the replies, but this poses no problem, we only require that the cookie is included in the request. The requests are sent inside HTML5 WebWorkers. This means our JavaScript code will run in the background of the browser, and any open page(s) stay responsive. We use GET requests, and carefully craft the values of our injected cookies so the targeted `auth` cookie is always at a fixed position in the keystream (modulo 256). Recall that this alignment is required to make optimal use of the Fluhrer-McGrew biases. An attacker can learn the required amount of padding by first letting the client make a request without padding. Since RC4 is a stream cipher, and no padding is added by the TLS protocol, an attacker can easily observe the length of this request. Based on this information it is trivial to derive the required amount of padding.

To test our attack in practice we implemented a tool in C which monitors network traffic and collects the necessary ciphertext statistics. This requires reassembling the TCP and TLS streams, and then detecting the 512-byte

(encrypted) HTTP requests. Similar to optimizing the generation of datasets as in Sect. 4.3.2, we cache several requests before updating the counters. We also created a tool to brute-force the cookie based on the generated candidate list. It uses persistent connections and HTTP pipelining [53, §6.3.2]. That is, it uses one connection to send multiple requests without waiting for each response.

In our experiments the victim uses a 3.1 GHz Intel Core i5-2400 CPU with 8 GB RAM running Windows 7. Internet Explorer 11 is used as the browser. For the server a 3.4 GHz Intel Core i7-3770 CPU with 8 GB RAM is used. We use nginx as the web server, and configured RC4-SHA1 with RSA as the only allowable cipher suite. This assures that RC4 is used in all tests. Both the server and client use an Intel 82579LM network card, with the link speed set to 100 Mbps. With an idle browser this setup resulted in an average of 4 450 requests per second. When the victim was actively browsing YouTube videos this decreased to roughly 4 100. To achieve such numbers, we found it's essential that the browser uses persistent connections to transmit the HTTP requests. Otherwise a new TCP and TLS handshake must be performed for every request, whose round-trip times would significantly slow down traffic generation. In practice this means the website must allow a `keep-alive` connection. While generating requests the browser remained responsive at all times. Finally, our custom tool was able to test more than 20 000 cookies per second. To execute the attack with a success rate of 94% we need roughly $9 \cdot 2^{27}$ ciphertexts. With 4 450 requests per seconds, this means we require 75 hours of data. Compared to the (more than) 2 000 hours required by AlFardan et al. [5, §5.3.3] this is a significant improvement. We remark that, similar to the attack of AlFardan et al. [5], our attack also tolerates changes of the encryption keys. Hence, since cookies can have a long lifetime, the generation of this traffic can even be spread out over time. With 20 000 brute-force attempts per second, all 2^{23} candidates for the cookie can be tested in less than 7 minutes.

We have executed the attack in practice, and successfully decrypted a 16-character cookie. In our instance, capturing traffic for 52 hours already proved to be sufficient. At this point we collected $6.2 \cdot 2^{27}$ ciphertexts. After processing the ciphertexts, the cookie was found at position 46 229 in the candidate list. This serves as a good example that, if the attacker has some luck, less ciphertexts are needed than our $9 \cdot 2^{27}$ estimate. These results push the attack from being on the verge of practicality, to feasible, though admittedly somewhat time-consuming.

4.7 Related Work

Due to its popularity, RC4 has undergone wide cryptanalysis. Particularly well known are the key recovery attacks that broke WEP [57, 163–165, 189]. Several other key-related biases and improvements of the original WEP attack have also been studied [82, 96, 126, 155].

We refer to Sect. 4.2.1 for an overview of various biases discovered in the keystream [5, 29, 56, 67, 75, 97, 99, 100, 124, 125, 127, 154]. In addition to these, the long-term bias $\Pr[Z_r = Z_{r+1} \mid 2 \cdot Z_r = i_r] = 2^{-8}(1 + 2^{-15})$ was discovered by Basu et al. [16]. While this resembles our new short-term bias $\Pr[Z_r = Z_{r+1}]$, in their analysis they assume the internal state $\mathcal{S}$ is a random permutation, which is true only after a few rounds of the PRGA. Isobe et al. searched for dependencies between initial keystream bytes by empirically estimating $\Pr[Z_r = y \wedge Z_{r-a} = x]$ for $0 \leq x, y \leq 255$, $2 \leq r \leq 256$, and $1 \leq a \leq 8$ [75]. They did not discover any new biases using their approach. Mironov modelled RC4 as a Markov chain and recommended to skip the initial $12 \cdot 256$ keystream bytes [111]. Paterson et al. generated keystream statistics over consecutive keystream bytes when using the TKIP key structure [124]. However, they did not report which (new) biases were present. Through empirical analysis, we show that biases between consecutive bytes are present even when using RC4 with random 128-bit keys.

The first practical attack on WPA-TKIP was found by Beck and Tews [164] and was later improved by other researchers [69, 168, 181, 185]. Recently several works studied the per-packet key construction both analytically [67] and through simulations [5, 124, 125]. For our attack we replicated part of the results of Paterson et al. [124, 125], and are the first to demonstrate this type of attack in practice. In [5] AlFardan et al. ran experiments where the two most likely plaintext candidates were generated using single-byte likelihoods [5]. However, they did not present an algorithm to return arbitrarily many candidates, nor extended this to double-byte likelihoods.

The SSL and TLS protocols have undergone wide scrutiny [5, 6, 24, 33, 46, 160]. Our work is based on the attack of AlFardan et al., who estimated that $13 \cdot 2^{30}$ ciphertexts are needed to recover a 16-byte cookie with high success rates [5]. We reduce this number to $9 \cdot 2^{27}$ using several techniques, the most prominent being usage of likelihoods based on Mantin's $AB\mathcal{S}AB$ bias [99]. Isobe et al. used Mantin's $AB\mathcal{S}AB$ bias, in combination with previously decrypted bytes, to decrypt bytes after position 257 [75]. However, they used a counting technique instead of Bayesian likelihoods. As a result, they require roughly 2^{34} ciphertexts to recover bytes 258 to 261 with high probability, while we require roughly 2^{30} ciphertexts. Additionally, it is not clear how resilient their technique is to errors in the decryption process when additional bytes would be decrypted. In [120] a

guess-and-determine algorithm combines $AB\mathcal{S}AB$ and Fluhrer-McGrew biases, requiring roughly 2^{34} ciphertexts to decrypt an individual byte with high success rates. Garman et al. targeted passwords sent over IMAP and BasicAuth [60]. They exploit biases in the initial keystream bytes of RC4, and combine them with a priori password distributions and password dictionaries. They obtain good success rates using roughly 2^{26} encryptions of a password, and estimate an attack in practice takes around 300 hours. Finally, Bricout et al. rigorously analyzed the structure and exploitation of Mantin's $AB\mathcal{S}AB$ bias [29]. Similar to our approach, they rely on known plaintext around the unknown targeted bytes. Additionally, they show how to estimate the rank of the correct 2-byte plaintext value. They used Viterbi and beam-search algorithms to generate a list of multi-byte plaintext candidates, though determining the rank of the correct multi-byte plaintext was left as an open problem.

4.8 Chapter Conclusion

While previous attacks against RC4 in TLS and WPA-TKIP were on the verge of practicality, our work pushes them towards being practical and feasible. After capturing $9 \cdot 2^{27}$ encryptions of a cookie sent over HTTPS, we can brute-force it with high success rates in negligible time. By running JavaScript code in the browser of the victim, we were able to execute the attack in practice within merely 52 hours. Additionally, by abusing RC4 biases, we successfully attacked a WPA-TKIP network within an hour. We consider it surprising this is possible using only known biases, and expect these types of attacks to further improve in the future. Based on these results, we strongly urge people to stop using RC4.

Chapter 5

Conclusion

> "Time passes, people move... Like a river's flow, it never ends... A childish mind will turn to noble ambition... The clear water's surface reflects growth... Now listen to the Serenade of Water to reflect upon yourself..."
>
> — *Shiek, The Legend of Zelda: Ocarina of Time*

We end this dissertation by summarizing our results, examining to what degree our initial goals have been fulfilled, and discussing open problems and interesting future research directions.

5.1 Summary of Contributions

Though WPA-TKIP was designed to be an intermediary solution, it is (still) supported by a large number of networks. We showed it fails to provide a sufficient level of security, by describing and demonstrating several new attacks. Moreover, it was also shown that TKIP can be attacked when used to encrypt broadcast and multicast packets. Since TKIP is used significantly more as a group cipher than as a unicast cipher, this demonstrates that weaknesses in TKIP are still of high practical value. However, this did not yet achieve our desired goal: many encrypted Wi-Fi networks continue to support TKIP.

At the same time we were able to implement several low-layer attacks against the Wi-Fi protocol using a commodity Wi-Fi device. This is surprising, since these

devices only offer a limited API to control the radio (e.g., we cannot transmit arbitrary signals, do not have access to raw signal data, cannot modify medium access algorithms, and so on). In particular we were able to implement a select jammer, which otherwise required expensive and state-of-the-art hardware.

While previous attacks against RC4 in TLS and WPA-TKIP were on the verge of practicality, our results push them towards being practical and feasible. After capturing roughly one billion encryptions of a cookie sent over HTTPS, we can brute-force it with high success rates in negligible time. By running JavaScript code in the browser of the victim, we were able to execute the attack in practice within merely 52 hours. Additionally, by abusing RC4 biases, we successfully attacked a WPA-TKIP network within an hour. Together with other recent attacks on RC4, these results motivated major browser vendors to drop support of RC4 [13, 86, 122]. In this regard, our work on RC4 has been a major success.

5.2 Future Work

The work presented in this dissertation can be further improved and extended. Some interesting research directions that can be worked out are discussed in this section.

5.2.1 Wireless Network Security

Our survey in April 2016 showed that more than half of all encrypted Wi-Fi networks still support WPA-TKIP. With this in mind, and given that the Wi-Fi Alliance has not prohibited usage of TKIP, it is worthwhile to search for new and more damaging attacks against this protocol. Existing attacks rely on MIC failure messages as an oracle to determine whether the ICV of a modified packet is correct. This makes attacks slow, and inapplicable against access points. An alternative route is finding other side channels that leak whether the ICV of a packet is correct. The first option that comes to mind is a timing-based side channel. However, these may be difficult to exploit in practice. Instead it may also be worthwhile to search for situations, where stations only react to management bits in the plaintext 802.11 header if the packet has a correct ICV.

During this work several vulnerabilities were discovered in implementations of WPA-TKIP and AES-CCMP (e.g., not verifying the authenticity of packets). Most of the time this functionality is offloaded to the network card, and implemented in closed source and proprietary firmware. Implementing a black-box test suite that systematically tests for such implementation vulnerabilities

might uncover additional flaws, and allows a systematic review of devices belonging to various vendors. Ideally, the test suite would be incorporated in the Wi-Fi Test Suite of the Wi-Fi Alliance [166], meaning only devices that pass the test suite receive the "Wi-Fi certified" logo.

5.2.2 Cryptanalysis of RC4

Our work uncovered a large set of novel short-term and long-term biases in the keystream of RC4. It is worthwhile to analytically determine their cause for two reasons. Naturally, it increases our understanding of RC4. Additionally, it may help in discovering more biases, which were missed due to the limited types of datasets (i.e., statistics) we were able to generate.

We only evaluated our plaintext recovery attacks under specific attack scenarios. For example, in our WPA-TKIP attack we decrypted a specific packet of a given length, and against TLS and HTTPS our focus was purely on decrypting 16-character cookies. Put differently, during our evaluation we mainly explored the influence of the number of captured ciphertexts on the success probability of our attacks. However, it is also worthwhile to determine the impact of other variables. More specifically, additional simulations can be performed for:

- Different lengths of the data that is being decrypted.
- Different character sets or restrictions on the plaintext (e.g. unrestricted, hexadecimal or ASCII characters only, base64-encoded, and so on).
- Different values for the maximum gap length of Mantin's $AB\mathcal{S}AB$ bias that we use.
- Situations where there is only known plaintext on one side of the data that we want to decrypt.

While performing these new simulations, it is also useful to keep track at which position the correct decrypted plaintext value was found (if found at all). This can be used to determine to which extent the number of generated candidates influence the probability of successfully decrypting certain data.

Currently our plaintext recovery attacks against RC4 only make use of the strongest known biases. Hence our attacks can be extended by also incorporating weaker biases. For instance, in the long-term setting where one keystream is continuously extended, the bias towards $(0, 0)$ and $(128, 0)$ at positions (Z_{w255}, Z_{w255+2}) can also be incorporated in our attacks. In principle our new long-term bias $Z_r = Z_{r+1}$ could also be included. However, due to its small

relative bias, we do not expect that this will significantly improve success rates. Another route for improving the attacks would be to take into account the application-specific character of the secret we are trying to decrypt. For instance, cookies generally only use a limited character set. Here it would be interesting to analyze common character sets and cookie lengths, and perform simulations to determine how much ciphertexts are needed to decrypt various types of cookies.

The attacks against RC4 presented in this dissertation combine biases in the keystream, to recover the value of a repeated secret. Put differently, our attacks do not work against secret information that is sent only once. However, if an adversary could recover the internal state of RC4, he would be able to decrypt all past and future data. Interestingly, there are known biases (correlations) between the keystream, and the internal state of RC4 [16, 77, 95]. This makes it interesting to search for a technique that combines all these correlations in an attempt to improve state recovery attacks.

5.3 Concluding Remarks

When looking back over the presented research, and taking into account related work and findings, an important general observation can be made. Namely, it is common that the impact of seemingly sophisticated attacks is underestimated. Though the first generation of certain attacks do seem impractical at times, they constantly evolve and improve, even in the face of widely deployed countermeasures. In other words, one should not ignore seemingly impractical attacks: they will only get better over time.

Another important aspect is that we were able to implement and demonstrate our attacks. This significantly strengthened the impact of our work. However, in the context of attacks against Wi-Fi networks, we must be vigilant [91, 174] that it remains possible to easily and cheaply implement proof-of-concepts. In particular, vendors are moving functionality away from open-source drivers, into closed-source and proprietary firmware [74], limiting our control over devices. More worrisome, the Federal Communications Commission (FCC) has recently proposed rule-making [50] that might cause vendors to lock down the functionality of wireless chips [153]. If this means vendors make it more difficult to monitor Wi-Fi traffic, or inject packets, it will become much harder to demonstrate the impact attacks. Hence it would also become more difficult to convince people to adopt more secure protocols, or patch existing implementations.

Bibliography

[1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, et al. "Imperfect forward secrecy: How Diffie-Hellman fails in practice". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM. 2015, pp. 5–17 (p. 8).

[2] AGD. *SSL and Export Ciphers: Logjam and FREAK*. Retrieved 25 April 2016 form `https://labs.portcullis.co.uk/blog/ssl-and-export-ciphers-logjam-and-freak/`. 2015 (p. 8).

[3] M. R. Albrecht and K. G. Paterson. *Lucky Microseconds: A Timing Attack on Amazon's s2n Implementation of TLS*. Cryptology ePrint Archive, Report 2015/1129. 2015 (pp. 7, 8).

[4] N. J. AlFardan. "On the Design and Implementation of Secure Network Protocols". PhD thesis. Royal Holloway, University of London, 2014 (p. 2).

[5] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. "On the Security of RC4 in TLS and WPA". In: *USENIX Security*. 2013 (pp. 10–12, 70–73, 75, 76, 85, 86, 91, 94, 97–101).

[6] N. J. AlFardan and K. G. Paterson. "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols". In: *IEEE Symposium on Security and Privacy*. 2013 (pp. 7, 8, 10, 71, 94, 97, 99, 101).

[7] *Amendment 6: Medium Access Control (MAC) Security Enhancements*. IEEE Std 802.11i. 2004 (pp. 5, 6).

[8] Anonymous (email id: nobody@jpunix.com). *Thank you Bob Anderson: RC4 Source Code*. Retrieved 19 April 2016 from `http://cypherpunks.venona.com/date/1994/09/msg00304.html`. Sept. 1994 (p. 10).

[9] N. Asokan, V. Niemi, and K. Nyberg. *Man-in-the-Middle in Tunnelled Authentication Protocols*. Cryptology ePrint Archive, Report 2002/163. 2002 (p. 9).

[10] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, et al. *DROWN: Breaking TLS using SSLv2*. Retrieved 26 April 2016 from `https://drownattack.com/drown-attack-paper.pdf`. 2016 (p. 9).

[11] G. V. Bard. "A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL". In: *SECRYPT*. INSTICC Press, 2006, pp. 99–109 (pp. 6, 8).

[12] R. Barnes, M. Thomson, A. Pironti, and A. Langley. *Deprecating Secure Sockets Layer Version 3.0*. RFC 7568. June 2015 (p. 5).

[13] R. Barnes. *Intent to ship: RC4 disabled by default in Firefox 44*. Retrieved 22 April 2016 from `https://groups.google.com/forum/#!topic/mozilla.dev.platform/JIEFcrGhqSM/discussion`. Sept. 2015 (pp. 70, 104).

[14] D. J. Barrett and R. E. Silverman. *SSH, the Secure Shell: The definitive guide*. O'Reilly Media, Inc., 2001 (p. 2).

[15] A. Barth. *HTTP State Management Mechanism*. RFC 6265. 2011 (p. 98).

[16] R. Basu, S. Ganguly, S. Maitra, and G. Paul. "A complete characterization of the evolution of RC4 pseudo random generation algorithm." In: *J. Mathematical Cryptology* 2.3 (2008), pp. 257–289 (pp. 77, 101, 106).

[17] E. Bayraktaroglu, C. King, X. Liu, G. Noubir, R. Rajaraman, and B. Thapa. "On the Performance of IEEE 802.11 under Jamming". In: *IEEE INFOCOM*. 2008 (p. 66).

[18] M. Beck. *Enhanced TKIP Michael Attacks*. Retrieved 4 Februari 2013 from `http://download.aircrack-ng.org/wiki-files/doc/enhanced_tkip_michael.pdf` (pp. 19, 26, 29, 32, 39).

[19] J. Bellardo and S. Savage. "802.11 denial-of-service attacks: real vulnerabilities and practical solutions". In: *Proc. of the 12th USENIX Security Symp.* 2003 (pp. 38, 66).

[20] D. Berbecaru and A. Lioy. "On the robustness of applications based on the SSL and TLS security protocols". In: *Public Key Infrastructure*. Lecture Notes in Computer Science. Springer, 2007, pp. 248–264 (p. 97).

[21] D. S. Berger, F. Gringoli, N. Facchi, I. Martinovic, and J. Schmitt. "Gaining Insight on Friendly Jamming in a Real-world IEEE 802.11 Network". In: *WiSec*. 2014 (pp. 43, 66).

[22] G. Berger-Sabbatel, A. Duda, O. Gaudouin, M. Heusse, and F. Rousseau. "Fairness and its Impact on Delay in 802.11 Networks". In: *GLOBECOM*. 2004 (p. 46).

[23] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. “A messy state of the union: Taming the composite state machines of TLS”. In: *IEEE Symposium on Security and Privacy (SP)*. 2015, pp. 535–552 (p. 8).

[24] K. Bhargavan, A. D. Lavaud, C. Fournet, A. Pironti, and P. Y. Strub. “Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS”. In: *IEEE Symposium on Security and Privacy (SP)*. IEEE. 2014, pp. 98–113 (pp. 7, 97, 101).

[25] K. Bhargavan, G. Leurent, D. Cadé, B. Blanchet, Z. Paraskevopoulou, C. Hriţcu, M. Dénès, L. Lampropoulos, B. C. Pierce, A. Delignat-Lavaud, et al. “Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH”. In: *NDSS*. 2016 (p. 8).

[26] K. Bicakci and B. Tavli. “Denial-of-Service attacks and countermeasures in IEEE 802.11 wireless networks”. In: *Comput. Stand. Interfaces* 31.5 (2009) (p. 38).

[27] A. Bittau, M. Handley, and J. Lackey. “The Final Nail in WEP’s Coffin”. In: *IEEE SP*. 2006 (pp. 5, 19, 26, 39).

[28] D. Bleichenbacher. “Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1”. In: *Advances in Cryptology (CRYPTO)*. Lecture Notes in Computer Science. Springer. 1998, pp. 1–12 (pp. 6, 9).

[29] R. Bricout, S. Murphy, K. G. Paterson, and T. van der Merwe. *Analysing and Exploiting the Mantin Biases in RC4*. Cryptology ePrint Archive, Report 2016/063. 2016 (pp. 75, 101, 102).

[30] E. Butler. “FireSheep: cookie snatching made simple”. In: *ToorCon Conference. San Diego, CA*. 2010 (p. 8).

[31] L. Butti and J. Tinnes. “Discovering and exploiting 802.11 wireless driver vulnerabilities”. In: *Journal in Computer Virology* 4.1 (2008), pp. 25–37 (p. 39).

[32] M. Cagalj, S. Ganeriwal, I. Aad, and J.-P. Hubaux. “On selfish behavior in CSMA/CA networks”. In: *IEEE INFOCOM*. 2005 (p. 66).

[33] B. Canvel, A. P. Hiltgen, S. Vaudenay, and M. Vuagnoux. “Password Interception in a SSL/TLS Channel”. In: *Advances in Cryptology (CRYPTO)*. Lecture Notes in Computer Science. 2003 (pp. 6, 8, 10, 71, 101).

[34] A. Cassola, W. Robertson, E. Kirda, and G. Noubir. “A Practical, Targeted, and Stealthy Attack Against WPA Enterprise Authentication”. In: *NDSS*. Apr. 2013 (pp. 9, 10, 43, 66).

[35] C. M. Chernick, C. Edington III, M. J. Fanto, and R. Rosenthal. *Guidelines for the selection and use of transport layer security (TLS) implementations*. US Department of Commerce, National Institute of Standards and Technology, 2005 (p. 11).

[36] R. Chirgwin. *RC4 crypto: Get RID of it already, say boffins*. Retrieved 23 April 2016 from `http://www.theregister.co.uk/2015/07/16/rc4_get_rid_of_it_already_say_boffins/`. July 2015 (p. 70).

[37] Cisco. *Cisco Wireless Controller Configuration Guide, release 8.0*. 2016 (p. 18).

[38] J. Daemen and V. Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer-Verlag, 2001 (pp. 4, 6).

[39] W. De Groef, D. Devriese, M. Vanhoef, and F. Piessens. "Information flow control for web scripts". In: *Foundations of Security Analysis and Design (FOSAD)*. Vol. 8604. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 124–145. ISBN: 9783319100814 (p. 124).

[40] *Dictionary Attack on Cisco LEAP Vulnerability*. Retrieved 26 April 2016 from `DictionaryAttackonCiscoLEAPVulnerability`. 2003 (p. 9).

[41] T. Dierks and C. Allen. *The TLS Protocol*. RFC 2246. June 1999 (p. 6).

[42] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.1*. RFC 4346. Apr. 2006 (pp. 7, 8).

[43] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. 2008 (pp. 7, 8, 71, 76).

[44] B. Dowling, M. Fischlin, F. Günther, and D. Stebila. "A cryptographic analysis of the TLS 1.3 handshake protocol candidates". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM. 2015, pp. 1197–1210 (p. 7).

[45] *DROWN Attack: SSLv2 has been known to be insecure for 20 years. What's the big deal?* Retrieved 16 April 2016 from `https://drownattack.com/#faq-whycare`. 2016 (p. 9).

[46] T. Duong and J. Rizzo. "Here come the XOR Ninjas". In: *Ekoparty Security Conference*. 2011 (pp. 6, 8, 10, 71, 94, 97, 99, 101).

[47] W. F. Ehrsam, C. H. Meyer, J. L. Smith, and W. L. Tuchman. *Message verification and transmission error detection by block chaining*. US Patent 4,074,066. Feb. 1978 (p. 4).

[48] Electronic Frontier Foundation (EFF). *The Message of Firesheep: "Baaaad Websites, Implement Sitewide HTTPS Now!"* Retrieved 26 April 2016 from urlhttps://www.eff.org/deeplinks/2010/10/message-firesheep-baaaad-websites-implement. 2010 (p. 7).

[49] D. Eppstein. “k-best enumeration”. In: *arXiv preprint arXiv:1412.5075* (2014) (p. 91).

[50] Federal Communications Commission (FCC). “Amendment of Parts 0, 1, 2, 15 and 18 of the Commission’s Rules regarding Authorization of Radiofrequency Equipment, Notice of Proposed Rule Making, ET Docket No. 15-170”. In: (July 2015) (p. 106).

[51] N. Ferguson. “Michael: an improved MIC for 802.11 WEP”. In: *IEEE doc. 802.11-2/020r0* (Jan. 2002) (pp. 19, 26, 39).

[52] Wi-Fi Alliance. *Technical Note: Removal of TKIP from Wi-Fi Devices*. Mar. 2015 (p. 18).

[53] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230. 2014 (p. 100).

[54] H. Finney. “An RC4 cycle that can’t happen”. In: *Post in Usenet sci.crypt* (1994) (p. 85).

[55] G. Fleishman. *Say Goodbye to WEP and TKIP*. Retrieved 26 November, 2012 from `http://wifinetnews.com/archives/2010/06/say_goodbye_to_wep_and_tkip.html`. 2010 (p. 33).

[56] S. R. Fluhrer and D. A. McGrew. “Statistical Analysis of the Alleged RC4 Keystream Generator”. In: *FSE*. Lecture Notes in Computer Science. 2000 (pp. 11, 74, 77–79, 85, 101).

[57] S. Fluhrer, I. Mantin, and A. Shamir. “Weaknesses in the key scheduling algorithm of RC4”. In: *SAC*. Lecture Notes in Computer Science. 2001 (pp. 5, 19, 76, 101).

[58] C. Fuchs and R. Kenett. “A Test for Detecting Outlying Cells in the Multinomial Distribution and Two-Way Contingency Tables”. In: *J. Am. Stat. Assoc.* 75 (1980), pp. 395–398 (p. 78).

[59] S. Ganu, K. Ramachandran, M. Gruteser, I. Seskar, and J. Deng. “Methods for Restoring MAC Layer Fairness in IEEE 802.11 Networks with Physical Layer Capture”. In: *REALMAN*. ACM, 2006 (pp. 47, 66).

[60] C. Garman, K. G. Paterson, and T. Van der Merwe. “Attacks only Get better: Password recovery Attacks against RC4 in TLS”. In: *USENIX Security*. 2015 (pp. 11, 12, 70, 71, 102).

[61] H. Gierow. *Der lange Abschied von RC4*. Retrieved 23 April 2016, form `http://www.golem.de/news/verschluesselung-der-lange-abschied-von-rc4-1507-114877.html`. July 2015 (p. 70).

[62] S. M. Glass and V. Muthukkumarasamy. “A Study of the TKIP Cryptographic DoS Attack”. In: *15th International Conference on Networks*. IEEE, 2007 (p. 38).

[63] Y. Gluck, N. Harris, and A. Prado. "BREACH: reviving the CRIME attack". In: *Black Hat Briefings*. 2013 (p. 9).

[64] O. Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2001. ISBN: 9780521791724 (pp. 3, 4).

[65] D. Goodin. *Once-theoretical crypto attack against HTTPS now verges on practicality*. Retrieved 23 April 2016 from `http://arstechnica.com/security/2015/07/once-theoretical-crypto-attack-against-https-now-verges-on-practicality/`. July 2015 (p. 70).

[66] M. Guennoun, A. Lbekkouri, A. Benamrane, M. Ben-Tahir, and K. El-Khatib. "Wireless networks security: Proof of chopchop attack". In: *WOWMOM*. IEEE, 2008 (p. 24).

[67] S. S. Gupta, S. Maitra, W. Meier, G. Paul, and S. Sarkar. *Dependence in IV-related bytes of RC4 key enhances vulnerabilities in WPA*. Cryptology ePrint Archive, Report 2013/476. 2013 (pp. 71, 75, 76, 93, 101).

[68] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. "Tool Release: Gathering 802.11n Traces with Channel State Information". In: *ACM SIGCOMM CCR* (2011) (p. 66).

[69] F. M. Halvorsen, O. Haugen, M. Eian, and S. F. Mjølsnes. "An Improved Attack on TKIP". In: *14^{th} Nordic Conf. on Secure IT Systems (NordSec)*. 2009 (pp. 39, 48, 67, 101).

[70] B. Harris. *Improved Arcfour Modes for the Secure Shell (SSH) Transport Layer Protocol*. RFC 4345. Jan. 2006 (p. 11).

[71] B. Harris and R. Hunt. "Review: TCP/IP security threats and attack methods". In: *Computer Communications* 22.10 (1999), pp. 885–897 (pp. 32, 94).

[72] J. Huang, J. Seberry, W. Susilo, and M. W. Bunder. "Security Analysis of Michael: The IEEE 802.11i Message Integrity Code". In: *EUC Workshops*. 2005, pp. 423–432 (pp. 19, 39).

[73] ICSI. *The ICSI Certificate Notary*. Retrieved 22 April 2016 from `http://notary.icsi.berkeley.edu` (p. 70).

[74] O. Ildis, Y. Ofir, and R. Feinstein. *Wardriving from your pocket*. REcon. 2013 (p. 106).

[75] T. Isobe, T. Ohigashi, Y. Watanabe, and M. Morii. "Full Plaintext Recovery Attack on Broadcast RC4". In: *FSE*. Lecture Notes in Computer Science. 2013 (pp. 71–74, 77, 101).

[76] jbyler. *Is it safe to enable SSLv2 ClientHello support?* Retrieved 26 April 2016 from `http://security.stackexchange.com/q/89930`. 2016 (p. 9).

[77] R. J. Jenkins. *ISAAC and RC4 (1996)*. Retrieved 26 February 2015 from `http://burtleburtle.net/bob/rand/isaac.html` (p. 106).

[78] E. Kasper. "Logjam, FREAK and Upcoming Changes in OpenSSL". In: *OpenSSL Blog*. 2015 (p. 8).

[79] K. Kaukonen and R. Thayer. *A stream cipher encryption algorithm "arcfour"*. 1999 (p. 11).

[80] Y. S. Kim, P. Tague, H. Lee, and H. Kim. "Carving secure wi-fi zones with defensive jamming". In: *ASIA CCS*. ACM, 2012, pp. 53–54 (p. 66).

[81] R. T. King Jr. "RSA Data Security Says Exposed Code Poses No Threat". In: *The Wall Street Journal* (Sept. 1994) (p. 11).

[82] A. Klein. "Attacks on the RC4 stream cipher". In: *Designs, Codes and Cryptography* 48.3 (2008), pp. 269–286 (p. 101).

[83] A. Kochut, A. Vasan, A. U. Shankar, and A. Agrawala. "Sniffing Out the Correct Physical Layer Capture Model in 802.11b". In: *ICNP*. 2004 (pp. 43, 47, 66).

[84] B. Könings, F. Schaub, F. Kargl, and S. Dietzel. "Channel switch and quiet attack: New DoS attacks exploiting the 802.11 standard". In: *LCN*. 2009 (p. 38).

[85] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. Feb. 1997 (p. 76).

[86] A. Langley. *Intent to deprecate: RC4*. Retrieved 22 April 2016 from `https://groups.google.com/a/chromium.org/forum/#!msg/security-dev/kVfCywocUO8/vgi_rQuhKgAJ`. Sept. 2015 (pp. 70, 104).

[87] A. Langley. *Overclocking SSL*. Retrieved 23 June 2016 from `https://www.imperialviolet.org/2010/06/25/overclocking-ssl.html`. 2010 (p. 7).

[88] A. Langley. *The POODLE bites again*. Retrieved 26 April 2016 from `https://www.imperialviolet.org/2014/12/08/poodleagain.html`. 2014 (p. 6).

[89] J. Lee, W. Kim, S.-J. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi. "An Experimental Study on the Capture Effect in 802.11a Networks". In: *Proc. of the 2nd ACM Intl. Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization*. ACM. 2007, pp. 19–26 (pp. 47, 60, 66).

[90] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolff. "A brief history of the Internet". In: *ACM SIGCOMM Computer Communication Review* 39.5 (2009), pp. 22–31 (p. 1).

[91] LibrePlanet. *Save WiFi*. Retrieved 26 April 2015, from `https://libreplanet.org/wiki/Save_WiFi` (p. 106).

[92] G. LosHuertos. "End of Privacy: Herding Firesheep in Starbucks". In: *CNNMoney guest columnist* (2010) (p. 7).

[93] lxgr. *Is RC4 a problem for password-based authentication?* Retrieved 24 April 2016 from `http://crypto.stackexchange.com/q/3451`. Aug. 2012 (p. 12).

[94] M. Lynn and R. Baird. "Advanced 802.11 Attack". In: *Black Hat Briefings*. 2002 (p. 67).

[95] S. Maitra and S. S. Gupta. "New long-term glimpse of RC4 stream cipher". In: *Information Systems Security*. Lecture Notes in Computer Science. Springer, 2013, pp. 230–238 (p. 106).

[96] S. Maitra and G. Paul. "New form of permutation bias and secret key leakage in keystream bytes of RC4". In: *FSE*. Lecture Notes in Computer Science. 2008 (p. 101).

[97] S. Maitra, G. Paul, and S. S. Gupta. "Attack on broadcast RC4 revisited". In: *Fast Software Encryption*. Lecture Notes in Computer Science. 2011 (pp. 73, 101).

[98] I. Mantin. "Bar Mitzvah Attack". In: *Black Hat Asia Briefings*. 2015 (p. 70).

[99] I. Mantin. "Predicting and Distinguishing Attacks on RC4 Keystream Generator". In: *EUROCRYPT*. Lecture Notes in Computer Science. 2005 (pp. 75, 77, 84, 101).

[100] I. Mantin and A. Shamir. "A Practical Attack on Broadcast RC4". In: *FSE*. Lecture Notes in Computer Science. 2001 (pp. 4, 73, 101).

[101] J. Manweiler, N. Santhapuri, S. Sen, R. Roy Choudhury, S. Nelakuditi, and K. Munagala. "Order Matters: Transmission Reordering in Wireless Networks". In: *MobiCom*. IEEE, 2009 (p. 60).

[102] J. Markoff. "A Secret Computer Code Is Out". In: *The New York Times* (Sept. 1994) (p. 11).

[103] M. Marlinspike. "New tricks for defeating SSL in practice". In: *Black Hat Briefings*. 2009 (p. 7).

[104] C. Matte, M. Cunche, R. Franck, and M. Vanhoef. "Defeating MAC Address Randomization Through Timing Attacks". In: *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '16)*. July 2016 (pp. 14, 123).

[105] N. McAllister. "Facebook: 'Don't worry, your posts are SECURE with us'". In: *The Register* (Aug. 2013) (p. 7).

[106] C. Meijer and R. Verdult. “Ciphertext-only cryptanalysis on hardened Mifare Classic cards”. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 18–30 (p. 2).

[107] M. Metzger. *KU Leuven PhD student has created virtually indefensible Wi-Fi stealth jamming attacks*. Retrieved 21 April 2016 from . Oct. 2015 (p. 42).

[108] P. E. Metzger. *Re: RC4*. Retrieved 20 April 2016 from `http://cypherpunks.venona.com/date/1994/09/msg00401.html`. Sept. 1994 (p. 11).

[109] S. K. Miller. “Facing the challenge of wireless security”. In: *IEEE Transactions on Computers* 34.7 (2001), pp. 16–18 (p. 5).

[110] B. Mills. *RC4 will no longer be supported in Microsoft Edge and IE11 [Updated]*. Retrieved 22 April 2016 from `https://blogs.windows.com/msedgedev/2016/03/16/rc4-will-no-longer-be-supported-in-microsoft-edge-and-ie11-beginning-in-april/`. Mar. 2016 (p. 70).

[111] I. Mironov. “(Not So) Random Shuffles of RC4”. In: *CRYPTO*. Lecture Notes in Computer Science. 2002 (pp. 74, 83, 101).

[112] V. Moen, H. Raddum, and K. J. Hole. “Weaknesses in the temporal key hash of WPA”. In: *Mobile Computing and Comm. Review* (2004) (pp. 48, 67).

[113] V. Moen, H. Raddum, and K. J. Hole. “Weaknesses in the temporal key hash of WPA”. In: *Mobile Computing and Communications Review* 8.2 (2004), pp. 76–83 (p. 39).

[114] B. Möller. *Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures*. Retrieved 26 April 2016 from `https://www.openssl.org/~bodo/tls-cbc.txt` (p. 6).

[115] B. Möller, T. Duong, and K. Kotowicz. *This POODLE bites: exploiting the SSL 3.0 fallback*. 2014 (p. 6).

[116] M. Morii and Y. Todo. “Cryptanalysis for RC4 and Breaking WEP/WPA-TKIP”. In: *IEICE Transactions* (2011), pp. 2087–2094 (pp. 23, 26, 38, 39, 48, 67).

[117] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. “You are what you include: Large-scale evaluation of remote JavaScript inclusions”. In: *CCS*. 2012, pp. 736–747 (p. 93).

[118] D. Nilsson and J. Goldberger. “Sequentially finding the N-Best List in Hidden Markov Models”. In: *International Joint Conferences on Artificial Intelligence*. 2001, pp. 1280–1285 (p. 91).

[119] G. Noubir, R. Rajaraman, B. Sheng, and B. Thapa. “On the Robustness of IEEE 802.11 Rate Adaptation Algorithms Against Smart Jamming”. In: *WiSec*. 2011, pp. 97–108 (p. 66).

[120] T. Ohigashi, T. Isobe, Y. Watanabe, and M. Morii. “Full Plaintext Recovery Attacks on RC4 Using Multiple Biases”. In: *IEICE Transactions* 98.1 (2015), pp. 81–91 (p. 101).

[121] T. Ohigashi and M. Morii. “A Practical Message Falsification Attack on WPA”. In: *Joint Workshop on Information Security (JWIS)*. 2009 (pp. 48, 66).

[122] A. Oot. *Ending support for the RC4 cipher in Microsoft Edge and Internet Explorer 11*. Retrieved 22 April 2016 from `https://blogs.windows.com/msedgedev/2015/09/01/ending-support-for-the-rc4-cipher-in-microsoft-edge-and-internet-explorer-11/`. Sept. 2015 (p. 104).

[123] S. Park, K. Kim, D. Kim, S. Choi, and S. Hong. “Collaborative QoS architecture between DiffServ and 802.11e wireless LAN”. In: *Vehicular Technology Conference*. 2003 (pp. 21, 27).

[124] K. G. Paterson, B. Poettering, and J. C. Schuldt. “Big Bias Hunting in Amazonia: Large-Scale Computation and Exploitation of RC4 Biases”. In: *AsiaCrypt*. Lecture Notes in Computer Science. 2014 (pp. 71, 75, 76, 79, 86, 91–93, 101).

[125] K. G. Paterson, J. C. N. Schuldt, and B. Poettering. “Plaintext Recovery Attacks Against WPA/TKIP”. In: *FSE*. Lecture Notes in Computer Science. 2014 (pp. 48, 67, 71, 75, 76, 92, 93, 101).

[126] G. Paul, S. Rathi, and S. Maitra. “On non-negligible bias of the first output byte of RC4 towards the first three bytes of the secret key”. In: *Designs, Codes and Cryptography* 49.1-3 (2008) (p. 101).

[127] S. Paul and B. Preneel. “A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher”. In: *FSE*. Lecture Notes in Computer Science. 2004 (pp. 74, 81, 84, 101).

[128] D. Pauli. *Researcher messes up Wi-Fi with an rPi and bargain buy radio stick*. Retrieved 21 April 2016 from . Oct. 2015 (p. 42).

[129] K. Pelechrinis, G. Yan, S. Eidenbenz, and S. Krishnamurthy. “Detecting Selfish Exploitation of Carrier Sensing in 802.11 Networks”. In: *IEEE INFOCOM*. 2009 (pp. 46, 66).

[130] J. A. Perry. *getting in trouble.*. Retrieved 19 April 2016 from `http://cypherpunks.venona.com/date/1994/09/msg00560.html`. Sept. 1994 (p. 10).

[131] A. Popov. *Prohibiting RC4 Cipher Suites*. RFC 7465. 2015 (p. 70).

[132] T. Pornin. *Re: Google is using RC4, but isn't RC4 considered unsafe?* Retrieved 26 April 2016 from `http://crypto.stackexchange.com/a/858`. 2011 (p. 12).

[133] T. Pornin. *Re: TLS: RC4 or not RC4?* Retrieved 26 April 2016 from `http://security.stackexchange.com/a/32498`. 2013 (p. 12).

[134] B. Preneel. "Mobile and Wireless Communications Security". In: *NATO ASI on Aspects of Network and Information Security*. 2008, pp. 119–133 (p. 5).

[135] O. Queseth. "The effect of selfish behavior in mobile networks using CSMA/CA". In: *Proc. of the 61st IEEE Vehicular Technology Conf.* 2005 (p. 66).

[136] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. 2014. URL: `http://www.R-project.org` (p. 78).

[137] L. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* (1989) (p. 90).

[138] S. Radosavac, J. S. Baras, and I. Koutsopoulos. "A framework for MAC protocol misbehavior detection in wireless networks". In: *Proc. of the 4th ACM workshop on Wireless security*. WiSe '05. 2005 (p. 66).

[139] M. Raya, J.-P. Hubaux, and I. Aad. "DOMINO: a system to detect greedy behavior in IEEE 802.11 hotspots". In: *MobiSys*. 2004, pp. 84–97 (pp. 46, 53, 54, 66).

[140] I. Ristic. *DROWN Abuses SSL v2 to Attack TLS*. Retrieved 26 April 2016 from `https://blog.qualys.com/securitylabs/2016/03/01/drown-abuses-ssl-v2-to-attack-rsa-keys-and-tls`. 2016 (p. 9).

[141] I. Ristić. *Is RC4 safe for use in SSL?* Retrieved 26 April 2016 from `http://blog.ivanristic.com/2009/08/is-rc4-safe-for-use-in-ssl.html`. 2009 (p. 12).

[142] I. Ristić. *Mitigating the BEAST attack on TLS*. Retrieved 24 June 2016 from `https://blog.qualys.com/ssllabs/2011/10/17/mitigating-the-beast-attack-on-tls`. 2011 (p. 8).

[143] R. L. Rivest and J. C. Schuldt. "Spritz—a spongy RC4-like stream cipher and hash function". In: *Proceedings of the Charles River Crypto Day, Palo Alto, CA, USA* 24 (2014) (p. 10).

[144] J. Rizzo and T. Duong. "The CRIME attack". In: *Ekoparty Security Conference*. 2012 (p. 9).

[145] P. Robyns, B. Bonné, P. Quax, and W. Lamotte. "Short paper: exploiting WPA2-enterprise vendor implementation weaknesses through challenge response oracles". In: *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. ACM. 2014, pp. 189–194 (p. 9).

[146] M. Roder and R. Hamzaoui. "Fast tree-trellis list Viterbi decoding". In: *Communications, IEEE Transactions on* 54.3 (2006), pp. 453–461 (p. 91).

[147] D. Roesler. *STUN IP Address requests for WebRTC*. Retrieved 17 June 2015 from `https://github.com/diafygi/webrtc-ips` (p. 96).

[148] RSA Data Security Inc. "RC4 Trademark". Serial no. 74463805. Nov. 1993 (p. 11).

[149] M. Safyan. *Bringing more secure search around the globe*. Retrieved 20 April 2016 from `https://search.googleblog.com/2012/03/bringing-more-secure-search-around.html`. Mar. 2012 (p. 7).

[150] B. Schneier. *Jamming Wi-Fi*. Retrieved 21 April 2016 from `https://www.schneier.com/blog/archives/2015/10/jamming_wi-fi.html`. Oct. 2015 (p. 42).

[151] B. Schneier. *New RC4 Attack*. Retrieved 21 April 2016 from `https://www.schneier.com/blog/archives/2015/07/new_rc4_attack_1.html`. July 2015 (p. 70).

[152] B. Schneier. *Re: RC4*. Retrieved 20 April 2016 from `http://cypherpunks.venona.com/date/1994/09/msg00394.html`. Sept. 1994 (p. 10).

[153] E. Schultz. "Yes, the FCC might ban your operating system". In: *Annual LibrePlanet Conference*. 2016 (p. 106).

[154] S. Sen Gupta, S. Maitra, G. Paul, and S. Sarkar. "(Non-)Random Sequences from (Non-)Random Permutations - Analysis of RC4 Stream Cipher". In: *Journal of Cryptology* 27.1 (2014), pp. 67–108 (pp. 73–75, 77, 83, 101).

[155] P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. "Discovery and exploitation of new biases in RC4". In: *SAC*. Lecture Notes in Computer Science. 2011 (p. 101).

[156] P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. "Statistical Attack on RC4 Distinguishing WPA". In: *EUROCRYPT*. 2011 (pp. 39, 67).

[157] N. Seshadri and C.-E. W. Sundberg. "List Viterbi decoding algorithms with applications". In: *IEEE Transactions on Communications* 42.234 (1994), pp. 313–323 (p. 91).

[158] Y. Sheffer, R. Holz, and P. Saint-Andre. *Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)*. RFC 7457. Feb. 2015 (p. 6).

[159] T. Shimomura. *Technical details of the attack described by Markoff in NYT*. Retrieved 23 April 2016 from `https://groups.google.com/d/msg/comp.security.misc/z5j323oQJeg/O7STL_6X_v4J`. Jan. 1995 (p. 2).

[160] B. Smyth and A. Pironti. "Truncating TLS Connections to Violate Beliefs in Web Applications". In: *WOOT'13: 7th USENIX Workshop on Offensive Technologies*. 2013 (pp. 97, 101).

[161] F. K. Soong and E.-F. Huang. "A tree-trellis based fast search for finding the n-best sentence hypotheses in continuous speech recognition". In: *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE. 1991, pp. 705–708 (p. 91).

[162] D. Sterndark. *RC4 Algorithm revealed*. Retrieved 19 April 2016 from `https://groups.google.com/d/msg/sci.crypt/TPS_Ix7aTJ4/oPwa0skAoxAJ`. Sept. 1994 (p. 10).

[163] A. Stubblefield, J. Ioannidis, and A. D. Rubin. "A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP)". In: *TISSEC* (2004), pp. 319–332 (pp. 19, 101).

[164] E. Tews and M. Beck. "Practical Attacks Against WEP and WPA". In: *WiSec*. ACM, 2009, pp. 79–86 (pp. 19, 22, 24–26, 28, 39, 47, 48, 64, 66, 75, 94, 101).

[165] E. Tews, R.-P. Weinmann, and A. Pyshkin. "Breaking 104 bit WEP in less than 60 seconds". In: *JISA*. 2007 (p. 101).

[166] The Wi-Fi Alliance. *Certification: Wi-Fi Test Suite*. `http://www.Wi-FiTestSuite.org` (p. 105).

[167] S. Thomas. *SSL and TLS essentials*. Wiley Computer Publishing, 2000, p. 3 (p. 1).

[168] Y. Todo, Y. Ozawa, T. Ohigashi, and M. Morii. "Falsification Attacks against WPA-TKIP in a Realistic Environment". In: *IEICE Transactions* (2012), pp. 588–595 (pp. 34, 39, 48, 67, 101).

[169] S. Turner and T. Polk. *Prohibiting Secure Sockets Layer (SSL) Version 2.0*. RFC 6176. Mar. 2011 (pp. 2, 5, 9).

[170] Twitter. *Securing your Twitter experience with HTTPS*. Retrieved 20 April 2016 from `https://blog.twitter.com/2012/securing-your-twitter-experience-with-https`. Feb. 2012 (p. 7).

[171] T. Van Goethem, P. Chen, N. Nikiforakis, L. Desmet, and W. Joosen. "Large-scale security analysis of the web: Challenges and findings". In: *Trust and Trustworthy Computing (TRUST)*. Springer, 2014, pp. 110–126 (p. 97).

[173] T. Van Goethem, M. Vanhoef, F. Piessens, and W. Joosen. "Request and Conquer: Exposing Cross-Origin Resource Size". In: *USENIX Security*. 2016 (p. 43).

[174] M. Vanhoef. *A Case For Open Radio Software*. 2015 (p. 106).

[175] M. Vanhoef. *Modwifi: Advanced Wi-Fi Attacks Using Commodity Hardware*. `http://modwifi.bitbucket.org/` (pp. 42, 44, 55, 58, 61, 65, 67).

[176] M. Vanhoef. *Modwifi: Advanced Wi-Fi Attacks Using Commodity Hardware*. `https://github.com/vanhoefm/modwifi` (p. 42).

[177] M. Vanhoef, W. De Groef, D. Devriese, F. Piessens, and T. Rezk. "Stateful declassification policies for event-driven programs". In: *Proceedings of the 27th IEEE Computer Security Foundations Symposium (CSF '14)*. IEEE, July 2014, pp. 293–307 (pp. 13, 124).

[178] M. Vanhoef, C. Matte, M. Cunche, L. Cardoso, and F. Piessens. *Tracking 802.11 stations without relying on the link layer identifier*. Retrieved 24 April 2016 from `http://ieee802.org/1/files/public/docs2016/e-cunche-dot11-tracking-0416.pdf` (p. 13).

[179] M. Vanhoef, C. Matte, M. Cunche, L. Cardoso, and F. Piessens. "Why MAC Address Randomization is not Enough: An Analysis of Wi-Fi Network Discovery Mechanisms". In: *Proceedings of the 11th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS '16)*. ACM, May 2016, pp. 413–424 (pp. 13, 123).

[181] M. Vanhoef and F. Piessens. "Advanced Wi-Fi attacks using commodity hardware". In: *ACSAC*. 2014 (p. 101).

[183] M. Vanhoef and F. Piessens. "All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS". In: *USENIX Security*. 2015 (p. 71).

[185] M. Vanhoef and F. Piessens. "Practical verification of WPA-TKIP vulnerabilities". In: *ASIA CCS*. ACM, 2013, pp. 427–436 (pp. 48, 67, 75, 101).

[187] M. Vanhoef and F. Piessens. "Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys". In: *USENIX Security*. 2016 (p. 43).

[188] S. Vaudenay. "Security Flaws Induced by CBC Padding—Applications to SSL, IPSEC, WTLS..." In: *Advances in Cryptology—EUROCRYPT 2002*. Lecture Notes in Computer Science. Springer. 2002, pp. 534–545 (pp. 6, 8).

[189] S. Vaudenay and M. Vuagnoux. “Passive–only key recovery attacks on RC4”. In: *SAC*. Lecture Notes in Computer Science. 2007 (p. 101).

[190] D. Whiting, R. Housley, and N. Ferguson. *Counter with CBC-MAC (CCM)*. RFC 3610. 2003 (pp. 4, 6).

[191] M. Wilhelm, I. Martinovic, J. B. Schmitt, and V. Lenders. “WiFire: A Firewall for Wireless Networks”. In: *SIGCOMM*. 2011 (p. 66).

[192] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Spec*. IEEE Std 802.11-2012. 2012 (pp. 5, 10, 19–21, 23, 24, 29, 36–38, 45, 47, 63, 71, 75, 76).

[193] A. Wool. “A note on the fragility of the Michael message integrity code”. In: *IEEE Transactions on Wireless Communications* 3.5 (2004), pp. 1459–1462 (pp. 19, 29, 39).

[194] J. Wright. “Weaknesses in LEAP challenge/response”. In: *DEF CON Hacking Conference*. 2003 (p. 9).

[195] W. Xu, W. Trappe, Y. Zhang, and T. Wood. “The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks”. In: *Proc. of ACM MobiHoc*. 2005, pp. 46–57 (p. 66).

List of Publications

Papers at International Conferences and Symposia, Published in Full in Proceedings

- M. Vanhoef and F. Piessens. "Predicting, Decrypting, and Abusing WPA2/802.11 Group Keys". In: *Proceedings of the 25th USENIX Security Symposium (USENIX Security '16)*. USENIX Association, Aug. 2016
- T. Van Goethem, M. Vanhoef, F. Piessens, and W. Joosen. "Request and Conquer: Exposing Cross-Origin Resource Size". In: *Proceedings of the 25th USENIX Security Symposium (USENIX Security '16)*. USENIX Association, Aug. 2016
- C. Matte, M. Cunche, R. Franck, and M. Vanhoef. "Defeating MAC Address Randomization Through Timing Attacks". In: *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '16)*. July 2016
- M. Vanhoef, C. Matte, M. Cunche, L. Cardoso, and F. Piessens. "Why MAC Address Randomization is not Enough: An Analysis of Wi-Fi Network Discovery Mechanisms". In: *Proceedings of the 11th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS '16)*. ACM, May 2016, pp. 413–424
- M. Vanhoef and F. Piessens. "All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS". in: *Proceedings of the 24th USENIX Security Symposium (USENIX Security '15)*. USENIX Association, Aug. 2015, pp. 97–112
- M. Vanhoef and F. Piessens. "Advanced Wi-Fi attacks using commodity hardware". In: *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC '14)*. ACM, Dec. 2014, pp. 256–265

- M. Vanhoef, W. De Groef, D. Devriese, F. Piessens, and T. Rezk. "Stateful declassification policies for event-driven programs". In: *Proceedings of the 27th IEEE Computer Security Foundations Symposium (CSF '14)*. IEEE, July 2014, pp. 293–307
- M. Vanhoef and F. Piessens. "Practical verification of WPA-TKIP vulnerabilities". In: *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS '13)*. ACM, May 2013, pp. 427–436

Chapters in Books

- W. De Groef, D. Devriese, M. Vanhoef, and F. Piessens. "Information flow control for web scripts". In: *Foundations of Security Analysis and Design (FOSAD)*. vol. 8604. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 124–145. ISBN: 9783319100814

"Wake me, when you need me."

— *Master Chief, Halo 3*

The End

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
IMINDS-DISTRINET
Celestijnenlaan 200A box 2402
B-3001 Leuven
http://www.cs.kuleuven.be