Statistical and Algebraic Cryptanalysis of Lightweight and Ultra-Lightweight Symmetric Primitives

THÈSE N^{O} 5415 (2012)

PRÉSENTÉE LE 27 AOÛT 2012

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE DE SÉCURITÉ ET DE CRYPTOGRAPHIE

PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Pouyan SEPEHRDAD

acceptée sur proposition du jury:

Prof. E. Telatar, président du jury
Prof. S. Vaudenay, directeur de thèse
Prof. A. Canteaut, rapporteur
Prof. A. Lenstra, rapporteur
Prof. W. Meier, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2012

To my two angels of God, my mother and father
whom without their help, devotion, affection, love and support
not only this dissertation could not be accomplished,
but I was not able to put my feet one step forward towards improvement.

Abstract

Symmetric cryptographic primitives such as block and stream ciphers are the building blocks in many cryptographic protocols. Having such blocks which provide provable security against various types of attacks is often hard. On the other hand, if possible, such designs are often too costly to be implemented and are usually ignored by practitioners. Moreover, in RFID protocols or sensor networks, we need lightweight and ultra-lightweight algorithms. Hence, cryptographers often search for a fair trade-off between security and usability depending on the application. Contrary to public key primitives, which are often based on some hard problems, security in symmetric key is often based on some heuristic assumptions. Often, the researchers in this area argue that the security is based on the confidence level the community has in their design. Consequently, everyday symmetric protocols appear in the literature and stay secure until someone breaks them. In this thesis, we evaluate the security of multiple symmetric primitives against statistical and algebraic attacks. This thesis is composed of two distinct parts:

In the first part, we investigate the security of RC4 stream cipher against statistical attacks. We focus on its applications in WEP and WPA protocols. We revisit the previous attacks on RC4 and optimize them. In fact, we propose a framework on how to deal with a pool of biases for RC4 in an optimized manner. During this work, we found multiple new weaknesses in the corresponding applications. We show that the current best attack on WEP can still be improved. We compare our results with the state of the art implementation of the WEP attack on Aircrack-ng program and improve its success rate. Next, we propose a theoretical key recovery and distinguishing attacks on WPA, which cryptographically break the protocol. We perform an extreme amount of experiments to make sure that the proposed theory matches the experiments. Finally, we propose a concrete theoretical and empirical proof of all our claims. These are currently the best known attacks on WEP and WPA.

In the second part, we shed some lights on the theory behind ElimLin, which is an algorithm for solving multivariate polynomial systems of equations. We attack PRESENT and LBlock block ciphers with ElimLin algorithm and compare their security using this algebraic technique. Then, we investigate the security of KATAN family of block ciphers and multi-purpose cryptographic primitive ARMADILLO against algebraic attacks. We break reduced-round versions of several members of KATAN family by proposing a novel pre-processing technique on the original algebraic representation of the cipher before feeding it to a SAT solver. Finally, we propose a devastating practical key recovery attack against the ARMADILLO1 protocol, which breaks it in polynomial time using a few challenge-response pairs.

Keywords: Symmetric cryptography, light-weight cryptography, statistical attacks, RC4 crypt-

analysis, WEP and WPA, block and stream ciphers, algebraic cryptanalysis, systems of sparse polynomial equations of low degree, challenge-response protocols, ElimLin, SAT solving techniques

Résumé

Les primitives cryptographiques à clé symétrique comme les systèmes de chiffrement par blocs et les systèmes de chiffrement à flot sont les composantes de base de nombreux protocoles cryptographiques. Il est souvent difficile d'avoir de telles composantes dont la sécurité est prouvée contre divers types d'attaques. Aussi, de telles conceptions sont souvent trop coûteuses pour être implémentées et sont habituellement ignorées en pratique. De plus, dans des environnements sous contraintes comme les protocoles RFID ou les réseaux de capteurs, il est primordial d'utiliser des algorithmes bas-coût. Il est donc nécessaire pour les cryptographes de rechercher un bon compromis entre la sécurité et l'utilisabilité. Ce compromis dépend de l'application. Contrairement aux primitives à clé publique qui sont souvent basées sur des problèmes supposés être difficiles à résoudre, la sécurité des systèmes à clé symétrique est souvent basée sur des suppositions heuristiques. Les chercheurs dans ce domaine affirment souvent que la sécurité est basée sur le niveau de confiance qu'a la communauté en leur construction. En conséquence, de nouveaux protocoles à clé symétrique apparaissent régulièrement et restent sûrs jusqu'à ce que quelqu'un les cassent. Dans cette thèse, nous évaluons la sécurité de plusieurs primitives à clé symétrique contre les attaques statistiques et algébriques. Cette thèse est composée de deux parties distinctes.

Dans la première partie, nous étudions la sécurité contre les attaques statistiques du système de chiffrement à flot RC4. Nous nous intéressons en particulier à ses applications dans les protocoles WEP et WPA. Nous revisitons les attaques développées précédemment sur RC4 et nous les optimisons. En fait, nous proposons un cadre pour traiter une banque de biais pour RC4 de façon optimisée. Dans ce travail, nous avons trouvé de nombreuses nouvelles vulnérabilités dans les applications correspondantes. Nous montrons aussi que la meilleure attaque actuellement existante sur WEP peut encore être améliorée. Nous comparons nos résultats avec le logiciel Aircrack-ng, la meilleure implémentation actuelle attaquant WEP et nous améliorons sa probabilité de succès. Ensuite, nous proposons des attaques par distingueurs ainsi qu'une attaque théorique sur WPA qui récupère la clé secrète. Ces attaques cassent le protocole d'un point de vue cryptographique. Nous effectuons une énorme quantité d'expériences afin d'être sûrs que la théorie que nous proposons soit bien vérifiée en pratique. Finalement, nous proposons des preuves théoriques et empiriques de toutes nos affirmations. Nos attaques sont actuellement les meilleures attaques sur WEP et WPA.

Dans la deuxième partie, nous apportons quelques éclaircissements sur la théorie derrière ElimLin qui est un algorithme utilisé pour résoudre des systèmes d'équations polynomiales multivariées. Nous étudions ensuite la sécurité contre les attaques algébriques de plusieurs systèmes de chiffrements par blocs et d'une primitive cryptographique multi-usage. Cette liste comprend KATAN, PRESENT et LBlock pour les chiffrement par blocs et ARMADILLO1 pour la primitive cryptographique multi-usage. Nous cassons des versions réduites de plusieurs membres de la famille KATAN

et nous proposons une nouvelle technique de prétraitement sur la représentation algébrique du système avant de le donner à un solveur SAT. Nous attaquons les systèmes de chiffrement par blocs PRESENT et LBlock en utilisant l'algorithme ElimLin et nous comparons leur sécurité en utilisant cette technique algébrique. Finalement, nous proposons une attaque à récupération de clé dévastatrice contre le protocole ARMADILLO1 qui le casse en temps polynomial en utilisant un faible nombre de paires challenge/réponse.

Mots-clés : Cryptographie à clé symétrique, cryptographie bas-coût, attaques statistiques, cryptanalyse de RC4, WEP et WPA, systèmes de chiffrement par blocs, systèmes de chiffrement à flot, systèmes d'équations polynomiales creuses de faible degré, protocoles de challenge/réponse, ElimLin, techniques résolvant SAT

Acknowledgment

There are always some influential people in anyone's academic life. Since this dissertation is the last written document in my student life, I would like to thank all those who changed my life and attitude during these exciting and challenging years:

I can not imagine how I can thank the best teachers of my life, my two angels of God, my parents, for all their support, help, sacrifices and love during my entire life; the **only** ones who were always there for me in both my happiness and hard time; the ones I love **the most** for ever and can not reciprocate even a small part of what they did for me.

At any period in my academic life, it was my dream to have someone beside me who has the answer to all my questions. I never had such an opportunity before I met Prof. Serge Vaudenay. For me, Serge is the realization of the *"Decryption Oracle"* in cryptography. I have never met anyone brighter and smarter in my life. One of the hardest decisions in my academic life was to decide whether to pursue PhD studies in England or to join Serge's group at EPFL, Switzerland. Quoting one of my favorite statements from one of the most influential people in Computer Science, Steve Jobs: "you can not connect the dots looking forward; you can only connect them looking backwards". Looking backward, coming to LASEC was the **best** decision I have ever made in my academic life. I would like to warmly thank Serge for giving me the opportunity to join his group, his help and support, providing such a nice environment for research and always motivating me to deliver precise results.

I would like to warmly thank Dr. Nicolas Courtois for his extreme kindness, help and support during the last 5 years. More importantly, he changed my attitude towards several points in life. I learned from him to follow my passion and to do research in areas which **I** find interesting, though other people believe what I am doing is of no interest. He was always motivating me to to set ambitious goals in my research and not to give up until I achieve them all.

I would also like to express my gratitude to Dr. Gwenaël Doërr together with Dr. Nicolas Courtois who encouraged me to pursue PhD studies at EPFL. I had one of the greatest time in my life studying at EPFL. They surely changed my life path.

Leaving Iran to study abroad was challenging and scary. I was not sure what types of issues I would encounter and I was not sure whether I am capable of handling them all. Facing a new culture, a new environment, handling pressure and tough deadlines and simultaneously being alone were not easy. I would like to thank Dr. Massoud Reza Hashemi for teaching me not to get scared of any new problem. He taught me to work hard until I can resolve the problems.

I would like to thank Prof. Anne Canteaut, Prof. Arjen Lenstra, Prof. Willi Meier and Prof.

Emre Telatar for having accepted to be part of the jury for my PhD defense. They also provided extremely constructive suggestions and comments.

I would like to thank Dr. Erik Tews for providing very useful comments on the Aircrack-ng software, during the implementation phase of our attacks on WEP. I would also like to thank Dr. Robert B Davies for providing extremely helpful comments on Generalized χ^2 distribution, used in the merging step of our attacks on WPA.

I would like to specifically thank great friends and colleagues: Dr. Jorge Nakahara, Petr Sušil, Dr. Martin Vuagnoux and Dr. Bingsheng Zhang for the time and effort they spent working with me in a team and their help and support during the last 4 years. I learned so many new things from them and I extremely enjoyed working with them.

I would like to express my gratitude to my best friend abroad, Shah Mahmood. I had such great time with him during my master and later during several visits in London. He was the reason I never felt lonely during my master. I will never forget his kindness during the last 5 years. I hope I had more friends like him.

I would like to thank Kamran Kalbasi for all his help, support and for being a great friend during the last 4 years.

I was always lucky with respect to office mates. Although they are very different, Rafik Chaabouni, Alexandre Duc and Dr. Khaled Ouafi are the best office mates I can imagine. I had such great time with all of them during my PhD. I hope one day, I can reciprocate all their help.

I would also like to thank all the (former and current) colleagues and friends at LASEC for providing such a nice and friendly environment at work: Dr. Jean-Philippe Aumasson, Asli Bay, Dr. Ioana Boureanu, Dr. Atefeh Mashatan, Dr. Katerina Mitrokotsa and Dr. Sylvain Pasini.

Finally, I would like to thank Martine Corval who was always very kind to me and never hesitated to help me with French and administrative issues. I really appreciate all her help and kindness.

Table of Contents

Chapter 1

Motivation

In the first part of this thesis, we perform extensive amount of analysis on RC4 stream cipher. We mainly focus on its applications in WEP and WPA protocols.

RC4 is the most widely used stream cipher in cryptography. Due to its simplicity, it has undergone an extreme amount of cryptanalysis since being anonymously disclosed in 1994. Indeed, it is widely used in various security protocols, such as SSL/TLS and Wi-Fi. In the following chapters, we are going to dig more into its security and particularly, we are going to exploit its weaknesses in WEP and WPA protocols.

WEP was already broken by Fluhrer, Mantin and Shamir (FMS) in 2001. Although their attack is practical, it is not fast enough. Moreover, since the introduction of FMS attack, almost all vendors restricted the class of weak keys used by the FMS attack. Therefore, that weakness can be avoided. Later, others such as Andreas Klein and a hacker called Korek came up with new weaknesses in RC4 and showed that the WEP key can be still recovered quite simply in practice. This work was further improved by Vuagnoux and Vaudenay and at the same time by Tews, Weinmann and Pyshkin (PTW). They showed that WEP can be broken in less than 60 seconds. That was the time when public softwares such as Aircrack-ng appeared which were implementing all the previous known attacks on WEP. Finally, Aircrack-ng was updated by Tews and Beck, using the refinement of all the previous attacks.

The drawback of all such analysis was that there was no concrete theory behind all the state of the art attacks, i.e., if a new weakness is found against RC4, it is not straightforward to compute how fast the ultimate algorithm would be. More clearly, there was no way to reproduce the results theoretically. Due to this lack of theory, the previous attacks in the literature set several of their parameters heuristically. For instance, Aircrack-ng implementers decided to use some heuristic settings for some parts of their attacks, gaining a better result in practice. Consequently, it was not clear at all how those attacks could be improved or whether there was any hope in finding an optimized technique.

In the following chapters, we introduce such a concrete theory. We compute all the parameters of our attacks theoretically and then, by performing an extensive amount of experiments, we certify all our theoretical analysis. Finally, we show that our attack is optimized. We improve the data and computational complexity of the attack on WEP and certify that it works in practice. This is achieved by proposing a theory on how to manipulate a pool of biases in RC4 in an optimized

fashion.

Using the same analysis, we also cryptanalyze the WPA protocol. For the first time, we provide a practical distinguisher for WPA. Then, we propose some temporary key recovery attacks. Although our key recovery attacks are not practical, they improve the state of the art attacks on this widely used wireless protocol. Currently, they are the best attacks against WPA. They also provide a better understanding of the RC4 stream cipher.

The results in the following chapters on RC4 are mainly derived by merging and extending our two papers published in SAC 2010 [SVV10] and EuroCrypt 2011 [SVV11] and another submission (extension of the previous two papers) under the review of Journal of Cryptology [SVV12], co-authored with Serge Vaudenay and Marin Vuagnoux.

In the second part of this thesis, we focus on algebraic techniques for cryptanalysis of symmetric primitives. First, we shed some light on the theory behind the ElimLin algorithm, which is an algorithm for solving polynomial systems of equations. We prove some fundamental theorems regarding this simple technique and compare it with other methods existing in the literature for solving multivariate polynomial systems of equations.

The idea behind the analysis of ElimLin was initiated by Nicolas Courtois and myself when I was doing my master thesis at University College London back in $2007 - 2008$. The main contribution of Chapter 10 which is Theorem 10.2 was introduced back in 2008 in [Sep08], but it was never formalized and rigorously proved until the present work. Here, in Chapter 10, we are providing a complete proof for such a surprising result on ElimLin. This fundamental result was published at FSE 2012 [SSV12], co-authored with Nicolas Courtois, Petr Sušil and Serge Vaudenay. Some of the attack simulations were also published in CANS 2009 [NSZW09], co-authored with Jorge Nakahara, Meiqin Wang and Bingsheng Zhang. That paper also evaluates the security of PRESENT block cipher against linear hulls and breaks 26 rounds out of 31 rounds of this lightweight block cipher.

Next, we propose a novel pre-processing approach applied to the algebraic representation of ciphers to make it appropriate for a SAT solver. Using this approach, we then attack reduced-round versions of KATAN family of block ciphers. This attack together with some more analysis (AIDA/cube and side channel attacks) were published in IndoCrypt 2010 [BCN+10], co-authored with Gregory Bard, Nicolas Courtois, Jorge Nakahara and Bingsheng Zhang.

Finally, we propose a devastating algebraic attack against the first version of ARMADILLO multi-purpose cryptographic primitive, proposed by our lab for Oridao company, which breaks the algorithm in practical complexity. In this thesis, we provide a polynomial time key recovery attack against the challenge-response application of this protocol. This result together with several variants of this protocol and the corresponding attacks were published in CHES 2010 and CARDIS 2011 [SSV11, BDN+11]. The first paper was co-authored with Petr Sušil and Serge Vaudenay. The second paper was co-authored with Stéphane Badel, Nilay Dagtekin, Jorge Nakahara, Khaled Ouafi, Nicolas Reffé, Petr Sušil and Serge Vaudenay.

The articles that were published and the ones which were submitted while pursuing the PhD program at EPFL (Oct 2008 - June 2012) are listed hereafter.

The List of Publications Included in This Thesis

Note. The paper entry 1 is an extended version for the Journal of Cryptology.

1. P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Tornado Attack on RC4 with Applications to WEP and WPA. (Submitted to Journal of Cryptology)
2. N. Courtois, P. Sepehrdad, P. Sušil, and S. Vaudenay. ElimLin Algorithm Revisited. In FSE'12.
3. P. Sepehrdad, P. Sušil, and S. Vaudenay. Fast Key Recovery Attack on ARMADILLO1 and Variants. In CARDIS'11, volume 7079, pages 133–150. Springer-Verlag, 2011.
4. P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Statistical Attack on RC4 - Distinguishing WPA. In EUROCRYPT'11, volume 6632, pages 343–363. Springer-Verlag, 2011.
5. S. Badel, N. Dağtekin, J. Nakahara, K. Ouafi, N. Reffé, P. Sepehrdad, P. Sušil, and S. Vaudenay. ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In CHES'10, volume 6225, pages 398–412. Springer-Verlag, 2010.
6. G.V. Bard, N. Courtois, J. Nakahara, P. Sepehrdad, and B. Zhang. Algebraic, AIDA/Cube and Side Channel Analysis of KATAN Family of Block Ciphers. In INDOCRYPT'10, volume 6498, pages 176–196. Springer-Verlag, 2010.
7. P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. New Key Recovery Attacks on RC4/WEP. In 27th Chaos Communication Congress (CCC), 2010.
8. P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Discovery and Exploitation of New Biases in RC4. In SAC'10, volume 6544, pages 74–91. Springer-Verlag, 2010.
9. J. Nakahara, P. Sepehrdad, B. Zhang, and M. Wang. Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT. In CANS'09, volume 5888, pages 58–75. Springer-Verlag, 2009.

The List of Publications Not Included in This Thesis

Note. The paper entry 2 in the list below obtained one of (out of 3) best paper awards of FSE 2012. The paper entry 1 is an extended version for the Journal of Cryptology.

1. M. Sajadieh, M. Dakhilalian, H. Mala, and P. Sepehrdad. Efficient Recursive Diffusion Layers for Block Ciphers and Hash Functions. (Submitted to Journal of Cryptology)
2. M. Sajadieh, M. Dakhilalian, H. Mala, and P. Sepehrdad. Recursive Diffusion Layers for Block Ciphers and Hash Functions. In FSE'12.
3. T. Baignères, P. Sepehrdad, and S. Vaudenay. Distinguishing Distributions Using Chernoff Information. In ProvSec'10, volume 6402, pages 144–165. Springer-Verlag, 2010.
4. J. Aumasson, J. Nakahara, and P. Sepehrdad. Cryptanalysis of the ISDB Scrambling Algorithm (MULTI2). In FSE'09, volume 5665, pages 296–307. Springer-Verlag, 2009.

Part I

Statistical Cryptanalysis of RC4 with Applications to WEP & WPA

Chapter 2

RC4 in WEP and WPA Protocols

2.1 Introduction

In this chapter, we are going to describe the RC4 stream cipher and its applications in IEEE 802.11 standard for wireless communication, i.e., WEP and WPA. Furthermore, we recall the previous cryptanalysis results against RC4 in both applications.

RC4 was designed by Rivest in 1987. It used to be a trade secret until it was anonymously posted on Cypherpunks mailing list in September 1994. Nowadays, due to its simplicity, RC4 is widely used in SSL/TLS, Microsoft Lotus, Oracle Secure SQL and Wi-Fi 802.11 wireless communications. The 802.11 [IEE03] used to be protected by WEP (Wired Equivalent Privacy) which is now being replaced by WPA (Wi-Fi Protected Access) due to security weaknesses.

WEP uses RC4 with a pre-shared key. Each packet is encrypted by an XOR to a keystream generated by RC4. The RC4 key is a pre-shared key prepended with a 3-byte nonce initialization vector IV. The IV is sent in clear for self-synchronization. There have been several attempts to break the full RC4 algorithm, but it has only been devastating so far in this scenario. Indeed, the adversary knows that the key is constant except the IV, which is known. An active adversary can even alter the IV. Nowadays, WEP is considered as being terribly weak, since passive attacks can recover the full key easily by assuming that the first bytes of every plaintext frame are known. This happens to be the case due to the protocol specifications.

In order to fix this problem, the Wi-Fi Alliance has replaced WEP by WPA [IEE03]. Peer authentication is based on IEEE 802.1X which accommodates a simple authentication mode based on a pre-shared key (WPA-PSK). Authentication creates a Temporary Key (TK). The TK then goes through the temporary key integrity protocol (TKIP) to derive the per-packet keys (PPK). The idea is that the TK is derived into a TTAK key to be used for a number of frames limited to 2^{16}. Each frame applies a simple transformation to the TTAK and a counter TSC to derive the RC4 per-packet key PPK. Again, the 3 first bytes of the RC4 key are known (they actually depend on a counter).

In addition to the key derivation, WPA provides a packet integrity protection scheme MIC which prevents from replaying or altering the IV. Thus, only passive key recovery attacks can be considered.

2.2 Description of RC4 and Notations

The RC4 stream cipher consists of two algorithms: the Key Scheduling Algorithm (KSA) and the Pseudo Random Generator Algorithm (PRGA). The RC4 engine has a state defined by two registers (words) i and j and one array (of N words) S defining a permutation over $\mathbf{Z}/N\mathbf{Z}$. The KSA generates an initial state for the PRGA from a random key K of L words as described in Figure 2.1. It starts with an array $\{0, 1, \ldots, N-1\}$, where $N = 2^8$ and swaps N pairs, depending on the value of the secret key K. At the end, we obtain the initial state $S'_0 = S_{N-1}$.

We define all the operators such as addition, subtraction and multiplication in the ring of integers modulo N represented as $\mathbf{Z}/N\mathbf{Z}$, or $\mathbf{Z}_N$, where $N = 256$ (i.e. *words* are *bytes*). Thus, $x + y$ should be read as $(x + y) \bmod N$.

KSA

1: **for** $i = 0$ to $N - 1$ **do**
2: $S[i] \leftarrow i$
3: **end for**
4: $j \leftarrow 0$
5: **for** $i = 0$ to $N - 1$ **do**
6: $j \leftarrow j + S[i] + K[i \bmod L]$
7: swap($S[i]$,$S[j]$)
8: **end for**

PRGA

1: $i \leftarrow 0$
2: $j \leftarrow 0$
3: **loop**
4: $i \leftarrow i + 1$
5: $j \leftarrow j + S[i]$
6: swap($S[i]$,$S[j]$)
7: output $z_i = S[S[i] + S[j]]$
8: **end loop**

Figure 2.1: The KSA and the PRGA algorithms of RC4.

Once the initial state S'_0 is created, it is used by the second algorithm of RC4, the PRGA. Its role is to generate a keystream of words of $\log_2 N$ bits, which will be XORed with the plaintext to obtain the ciphertext. Thus, RC4 computes the loop of the PRGA each time a new keystream word z_i is needed, according to the algorithm in Figure 2.1. Note that each time a word of the keystream is generated, the internal state (i, j, S) of RC4 is updated.

Sometimes, we consider an idealized version RC4$^\star(t)$ of RC4 defined by a parameter t as shown in Figure 2.2. Namely, after round t, j is assigned randomly. This model has been already used in the literature such as in [Max05, Roo95, PM07].

Let $S_i[k]$ (resp. $S'_i[k]$) denote the value of the permutation defined by the array S at index k, after the round i in the KSA (resp. the PRGA). We also denote $S_{N-1} = S'_0$. Let j_i (resp. j'_i) be the value of j after round i of the KSA (resp. the PRGA), where the rounds are indexed with respect to i. Thus, the KSA has rounds $0, 1, \ldots, N-1$ and the PRGA has rounds $1, 2, \ldots$. The KSA and

KSA$^\star(t)$

1: **for** $i = 0$ to $N - 1$ **do**
2: $S[i] \leftarrow i$
3: **end for**
4: $j \leftarrow 0$
5: **for** $i = 0$ to $N - 1$ **do**
6: **if** $i \leq t$ **then**
7: $j \leftarrow j + S[i] + K[i \bmod L]$
8: **else**
9: $j \leftarrow$ random
10: **end if**
11: swap($S[i]$,$S[j]$)
12: **end for**

PRGA*

1: $i \leftarrow 0$
2: $j \leftarrow 0$
3: **loop**
4: $i \leftarrow i + 1$
5: $j \leftarrow$ random
6: swap($S[i]$,$S[j]$)
7: output $z_i = S[S[i] + S[j]]$
8: **end loop**

Figure 2.2: The KSA$^\star(t)$ and the PRGA* algorithms of RC4$^\star(t)$.

the PRGA are defined by

KSA

$$\begin{aligned} j_{-1} &= 0 \\ j_i &= j_{i-1} + S_{i-1}[i] + K[i \bmod L] \\ S_{-1}[k] &= k \\ S_i[k] &= \begin{cases} S_{i-1}[j_i] & \text{if } k = i \\ S_{i-1}[i] & \text{if } k = j_i \\ S_{i-1}[k] & \text{otherwise} \end{cases} \end{aligned}$$

PRGA

$$\begin{aligned} j'_0 &= 0 \\ j'_i &= j'_{i-1} + S'_{i-1}[i] \\ S'_0[k] &= S_{N-1}[k] \\ S'_i[k] &= \begin{cases} S'_{i-1}[j'_i] & \text{if } k = i \\ S'_{i-1}[i] & \text{if } k = j'_i \\ S'_{i-1}[k] & \text{otherwise} \end{cases} \\ z_i &= S'_i[S'_i[i] + S'_i[j'_i]] \end{aligned}$$

2.3 Description of WEP

WEP [IEE99b] uses a 3-byte IV concatenated to a secret key of 40 or 104 bits (5 or 13 bytes) as an RC4 key. Thus, the RC4 key size is either 64 or 128 bits. This is because the packets can be simply lost through the wireless channel due to a transmission error. So, all the packets should be encrypted independently. Since the RC4 design does not accept an IV by default, WEP generates a per packet key for each packet. A devastating problem of WEP is that the 13 bytes of the key do not change for each packets encryption, while the first 3 bytes of the key are changing. Thus, the attacker can run a statistical attack on the key. This was avoided in WPA. In this thesis, we do not consider the 40-bit key variant. So, $L = 16$. In fact, we have

$$K = K[0]\|K[1]\|K[2]\|K[3]\|\cdots\|K[15] = \mathsf{IV}_0\|\mathsf{IV}_1\|\mathsf{IV}_2\|K[3]\|\cdots\|K[15]$$

where IV_i represents the $(i+1)$-th byte of the IV and $K[3]\|...\|K[15]$ represents the fixed secret part of the key. In theory, the value of the IV should be random but in practice, it is a counter, mostly in little-endian and it is incremented by one each time a new 802.11b frame is encrypted. Sometimes, some particular values of the IV are skipped to thwart the specific attacks based on "weak IV's". Thus, each packet uses a slightly different key.

To protect the integrity of the data, a 32-bit long CRC32 check sum called ICV is appended to the data. Similar to other stream ciphers, the resulting stream is XORed with the RC4 keystream and it is sent through the communication channel together with the IV in clear. On the receiver's end, the ciphertext is again XORed with the shared key and the plaintext is recovered. It checks the linear error correcting code and it either accepts the data or declines it.

It is well known [PR88, TWP07, VV07] that a relevant portion of the plaintext is practically constant and that some other bytes can be predicted. They correspond to the LLC header and the SNAP header and some bytes of the TCP/IP and ARP encapsulated frames. For example, by XORing the first byte of the ciphertext with the constant value 0xAA, we obtain the first byte of the keystream. Thus, even if these attacks are called known plaintext attacks, they are ciphertext only in practice (see Figure 2.3).

The attacker eavesdrops the ARP packets in the network and since the plaintext bytes are known up to the 32-nd byte, she can compute $z_1, \ldots, z_{32}$ values using the ciphertext. It is also possible to inject data into the network in an active attack. Because the ARP replies expire quickly, it usually takes only a few seconds or minutes until an attacker can capture an ARP request and start re-injecting it [TWP07]. The first public implementation of a practical re-injection attack was in the BSD − Airtools package [Hul]. Moreover, the TCP packets can be used as well, but some of the data frames are not known in this case. We will see later that the Klein and the Maitra-Paul attacks require z_i and z_{i+1} to recover $\bar{K}[i]$ and $\bar{K}[i+1]$ respectively. Hence, in reality we are not able to use those attacks to recover some bytes of the key. This is not the case for the Korek attacks, since they only require z_1 and z_2. Thus, if the TCP packets are used, more of them are needed to launch a successful attack.

2.4 Description of WPA

Since WEP was shown to be terribly weak, the 802.11 WG suggested a new protocol called Wi-Fi Protected Access (WPA). Its goal is to resolve all the security problems in WEP. Since WEP was already implemented in many commercial products, it was not cost effective to discard the protocol and design a new one from scratch. A dirty fix was to have a software patch on the existing hardware. In fact, the key is scrambled in WPA using a key mixing function and then it is given to RC4. Hence, WPA uses a completely different key for each packet. This was not the case for WEP.

WPA includes a key hashing function [HWF02] to defend against the Fluhrer, Mantin and Shamir attack [FMS01]. It also includes a Message Integrity Code (MIC) [Fer02] to provides integrity and a key management scheme based on 802.1X [WG.01] to avoid the key reuse and to ease the key distribution.

The 128-bit Temporal Key (TK) is a per-session key. It is derived from the key management scheme during the authentication and is given as an input to the phase1 key hashing function (key mixing algorithm) together with the 48-bit Transmitter Address (TA) and a 48-bit TKIP Sequence Counter (TSC) which is sometimes called an IV. We will avoid this latter name to avoid confusion with the first 3 bytes of the RC4 key (which indeed only depend on the TSC, but with a shorter length).

To maintain the integrity of the data, a non-linear message integrity function called MIC is used. It takes a MIC key, the TA, the receiver address and the message as the input and it

ARP Packet	
0xAA	DSAP
0xAA	SSAP
0x03	CTRL
0x00 0x00 0x00	ORG Code
0x08 0x06	ARP
0x00 0x01	Ethernet
0x08 0x00	IP
0x06	Hardware size
0x04	Protocol
0x00 0x??	Opcode Request/Reply
0x?? 0x?? 0x?? 0x?? 0x?? 0x??	MAC addr src
0x?? 0x?? 0x?? 0x??	IP src
0x?? 0x?? 0x?? 0x?? 0x?? 0x??	MAC addr dst
0x?? 0x?? 0x?? 0x??	IP dst

TCP Packet	
0xAA	DSAP
0xAA	SSAP
0x03	CTRL
0x00 0x00 0x00	ORG Code
0x08 0x00	IP
0x45	IP Version + Header length
0x?? 0x??	Packet length
0x?? 0x??	IP ID RFC815
0x??	Fragment type and offset
0x??	TTL
0x06	TCP type
0x?? 0x??	Header checksum
0x?? 0x?? 0x?? 0x??	IP src
0x?? 0x?? 0x?? 0x??	IP dst
0x?? 0x??	Port src
0x?? 0x??	Port dst

Figure 2.3: The plaintext bytes of the 802.11 data frames encapsulating ARP and TCP protocols [VV07]. The values in white are almost fixed or can be computed dynamically. The values in light Grey can be guessed. The values in dark Grey are not predictable.

outputs the message concatenated with a MIC-tag (see [Fer02] for the MIC description). The entire encapsulation process is depicted in Figure 2.4. As can be seen, WPA is a wrapper for WEP.

The TK can be used to encrypt up to 2^{48} packets. Every packet has a 48-bit index TSC which is split into IV32 and IV16. The IV32 counter is incremented every 2^{16} packets. The packet is encrypted using a 128-bit RC4KEY which is derived from the TK, the TSC and some other parameters (for example, the device addresses) which can be assumed constant and known by the adversary for our purpose. As for WEP, the first three bytes of the RC4KEY only depend on the TSC so they are not secret. The derivation works in two phases. The first phase does not depend on the IV16 and is done once every 2^{16} packets for efficiency reasons. It derives a 80-bit key TTAK, called the TKIP-mixed Transmit Address and Key (TTAK) in the standard (but denoted the P1K in the reference code):

$$\mathsf{TTAK} = \mathsf{phase1}(\mathsf{TK}, \mathsf{TA}, \mathsf{IV32})$$

phase1 consists of two steps. It is depicted in Algorithm 2.1. The second phase uses the TK and the IV16 to derive a 96-bit key PPK which is then turned into the RC4KEY:

$$\mathsf{RC4KEY} = \mathsf{phase2}(\mathsf{TK}, \mathsf{TTAK}, \mathsf{IV16})$$

phase2 consists of three steps. It is depicted in Algorithm 2.2.

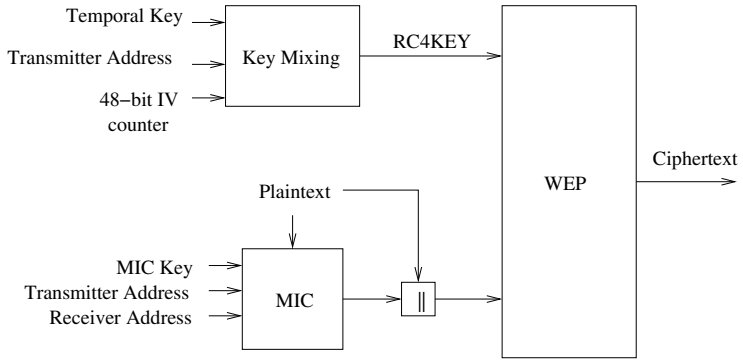


Figure 2.4: The WPA encapsulation process [MRH04].

The S[.] function is a 16×16 bijective non-linear S-box, defined by a look up table in [HWF02]. We also define $\mathsf{Mk16}(X, Y) = X \parallel Y$. Addition is done modulo 2^{16}. The functions $\mathsf{low16}(x)$ and $\mathsf{high16}(x)$ extract the 16 LSB and 16 MSB of x. It is similar for $\mathsf{low8}(x)$ and $\mathsf{high8}(x)$. Finally, RotR1 represents rotation to the right by 1. The key derivation of WPA based on a pre-shared key is depicted on Figure 2.5 (without the protocol parameters such as the TA).

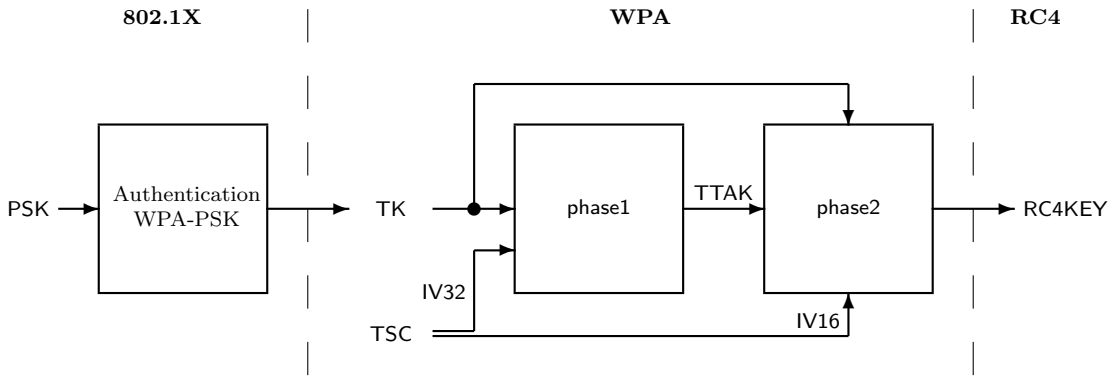


Figure 2.5: The WPA key derivation based on a pre-shared key authentication method.

In what follows, we denote $K[i] = \mathsf{RC4KEY}[i \bmod 16]$ and $\mathsf{IV} = K[0] \| K[1] \| K[2]$ to use the same notations as in WEP. By convention, TTAK and PPK are considered as vectors or 16-bit words. The TK and the RC4KEY are considered as vectors or 8-bit words. Vectors are numbered starting from 0.

Note that a filter avoids the use of some weak IV classes. Actually, it avoids only the weak class of IV's discovered by Fluhrer, Mantin, and Shamir [FMS01].

Algorithm 2.1 The phase1 of the WPA key hashing function.

phase1, step1:

1: $\mathsf{TTAK}[0] = \mathsf{low16}(\mathsf{IV32})$
2: $\mathsf{TTAK}[1] = \mathsf{high16}(\mathsf{IV32})$
3: $\mathsf{TTAK}[2] = \mathsf{Mk16}(\mathsf{TA}[1], \mathsf{TA}[0])$
4: $\mathsf{TTAK}[3] = \mathsf{Mk16}(\mathsf{TA}[3], \mathsf{TA}[2])$
5: $\mathsf{TTAK}[4] = \mathsf{Mk16}(\mathsf{TA}[5], \mathsf{TA}[4])$

phase1, step2:

6: **for** $i = 0$ to 7 **do**
7: $\quad j = 2 * (i \ \& \ 1)$
8: $\quad \mathsf{TTAK}[0] = \mathsf{TTAK}[0] + \mathsf{S}[\mathsf{TTAK}[4] + \mathsf{Mk16}(\mathsf{TK}[j+1], \mathsf{TK}[j])]$
9: $\quad \mathsf{TTAK}[1] = \mathsf{TTAK}[1] + \mathsf{S}[\mathsf{TTAK}[0] + \mathsf{Mk16}(\mathsf{TK}[j+5], \mathsf{TK}[j+4])]$
10: $\quad \mathsf{TTAK}[2] = \mathsf{TTAK}[2] + \mathsf{S}[\mathsf{TTAK}[1] + \mathsf{Mk16}(\mathsf{TK}[j+9], \mathsf{TK}[j+8])]$
11: $\quad \mathsf{TTAK}[3] = \mathsf{TTAK}[3] + \mathsf{S}[\mathsf{TTAK}[2] + \mathsf{Mk16}(\mathsf{TK}[j+13], \mathsf{TK}[j+12])]$
12: $\quad \mathsf{TTAK}[4] = \mathsf{TTAK}[4] + \mathsf{S}[\mathsf{TTAK}[3] + \mathsf{Mk16}(\mathsf{TK}[j+1], \mathsf{TK}[j])] + i$
13: **end for**

2.5 State of the Art Attacks

2.5.1 Previous Attacks on RC4

We consider three approaches for the cryptanalysis of RC4: attacks based on the weaknesses of the Key Scheduling Algorithm (KSA), attacks based on the weaknesses of the Pseudorandom Generator Algorithm (PRGA) and the black box analysis.

As for the KSA, one of the first weaknesses published on RC4 was discovered by Roos [Roo95] in 1995. This correlation binds the secret key bytes to the initial state S'_0. Roos [Roo95] and Wagner [Wag95] identified classes of weak keys which reveal the secret key if the first key bytes are known. This property has been largely exploited to break WEP (see [Bit03, FMS01, Hul01, Kor04b, Kor04a, SVV10, BT09, TWP07, VV07]). Another class of results concerns the inversion problem of the KSA: given the final state of the KSA, the problem is to recover the secret key [BC08, PM07].

Regarding the PRGA, the analysis has been largely motivated by distinguishing attacks [FM01, Gol97, Man05b, Max05] or the initial state reconstruction from the keystream bytes [Gol00, KMP$^+$98, MK08, TBNT07] with the complexity of 2^{241} for the best state recovery attack. Relevant studies of the PRGA reveal biases in the keystream output bytes in [MS01, PP04]. Mironov recommends in [Mir02] that the first 512 initial keystream bytes must be discarded to avoid these weaknesses.

Jenkins published in 1996 two biases in the PRGA of RC4 on his website [Jen96]. These biases have been generalized by Mantin in his Master Thesis [Man01]. In 2008, Paul, Rathi and Maitra [PRM08] discovered a biased output index of the first keystream word generated by the PRGA. Another bias on the PRGA has been discovered by Maitra and Paul in [MP08].

In practice, key recovery attacks on RC4 must bind the KSA and the PRGA weaknesses to correlate

secret key words to the keystream words. Some biases on the PRGA [Kle08, PRM08, MP08] have been successfully bound to the Roos correlation [Roo95] to provide known plaintext attacks. Another approach is the black box analysis, which does not require any binding. This approach will be elaborated later.

In [SPSG11], some biases in the initial keystream bytes of RC4 were found. The authors also showed that the value of j_2 tends towards 4. Finally, they extended the attack of Mantin and Shamir on Broadcast RC4 [MS01] to recover the bytes 3 to 255 of the plaintext given N^3 ciphertexts. Finally, in [SGMPS11b], the authors provided a theoretical justification for the probability distribution of the first keystream byte z_1 of RC4.

In the following chapters, we are going to elaborate some new empirical biases on RC4. Later, some of these biases were proved in [SGMPS11a]. Using one of these biases, the authors mounted a key length discovery attack on RC4.

2.5.2 Previous Attacks on WEP

Regarding the application of RC4 in the WEP protocol, some of the serious security flaws in WEP were discovered by Borisov, Goldberg and Wagner [BGW01]. They showed that WEP fails to achieve its security goals. The 802.11 standard does not necessarily ask for the IV to be changed with every packet, so an implementation has no obligation to reuse the same IV for all the packets. Since RC4 is a stream cipher, this means that we can find two packets encrypted with exactly the same keystream if the IV is reused. This leaks the XOR of the corresponding two packets. Many such pairs can be gathered in matter of hours. In fact, there are many frequency analysis techniques in the literature (see for instance [Sin99]) to derive the corresponding packets. They also showed that the ICV is not cryptographically secure. In fact, CRC is a linear function of the message and it was shown in [BGW01] that it is possible to make a controlled modification of the ciphertext without disrupting the check sum. This means that WEP does not even provide integrity. Finally, the 802.11 standard does not specify how the distribution of the keys is to be accomplished.

WEP key recovery process is harder in practice than in theory. Indeed, some bytes of the keystream may be unknown. Moreover, theoretical success probability has often been miscalculated and conditions to recover the secret key are not the same depending on the paper. For example, [TWP07, VV07, BT09, SVV10] check 10^6 most probable keys instead of the first one as in [FMS01, Kor04b, Kor04a, Kle08, SIR02, SIR04]. Additionally, IEEE 802.11 standard does not specify how the IV's should be chosen. Thus, some attacks consider randomly picked IV's or incremental IV's (both little-endian and big-endian encoded). Some implementations specifically avoid some classes of IV's which are weak with respect to some attacks.

To unify the results, during the entire thesis, we consider recovering a random 104-bit long secret key with random IV's. This corresponds to the default IV behavior of the 802.11 GNU/Linux stack. We compare the previous and the new results using both theoretical and practical analysis.

– In [FMS01], Fluhrer, Mantin and Shamir's attack is only theoretically described. The authors postulate that 4 million packets would be sufficient to recover the secret key of WEP with success probability of 50% with incremental IV's. A practical implementation of this attack has been realized by Stubblefield, Ioannidis and Rubin [SIR02, SIR04]. They showed that between 5 million to 6 million packets are needed to recover the secret key using the FMS

attack. Note that in 2001, almost all wireless cards were using incremental IV's in big-endian.

- There is no theoretical analysis of the Korek [Kor04a, Kor04b] key recovery attacks. Only practical implementations such as Aircrack-ng [DO11] are available. Additionally, Aircrack-ng classifies the most probable secret keys and does a brute-force attack on them. The success probability of 50% is obtained when about 100 000 packets are captured with random IV's. Note that the amount of the brute-forced keys depends on the values of the secret key and the *"Fudge"* factor (the number of trials on the key), a parameter chosen by the attacker. By default, around 1 000 to 1 000 000 keys are brute-forced. We will improve the conditions of the Korek attacks and prove the success probability of all such biases in the following chapters.

- In [Kle08], Klein showed theoretically that his new attack needs about 25 000 packets with random IV's to recover the secret key with probability 50%. Note that, there is no practical implementation of the Klein attack, but both PTW [TWP07] and VV07 [VV07] attacks, which theoretically improve the key recovery process, need more than 25 000 packets. So, the theoretical success probability of the Klein attack was over estimated. We implemented this attack and we obtained the success probability of 50% with about 60 000 packets (random IV's).

- Physkin, Tews and Weinmann showed in [TWP07] that the secret key can be recovered with only 40 000 packets for the same success probability (random IV's). However, this attack brute-forces the 10^6 most probable secret keys. Thus, the comparison with previous attacks is less obvious. Moreover, there is no theoretical analysis of this attack, only practical results are provided by the authors. We confirmed this practical result.

- Vaudenay and Vuagnoux [VV07] showed an improved attack, where the same success probability can be reached with an average of 32 700 packets with random IV's. This attack also tests the 10^6 most probable secret keys. Moreover, only practical results are provided by the authors. We confirmed this practical result.

- According to [BT09], Beck and Tews re-implemented the [VV07] attack in 2009, obtaining the same success probability with only 24 200 packets using Aircrack-ng in "interactive mode". Using this strategy, much less number of packets is required. No other previous attack used this strategy. The 10^6 most probable secret keys are brute-forced. Note that we were not able to reproduce this result. We could successfully recover the secret key with around 30 000 packets only with random IV's and Aircrack-ng in non-interactive mode to be able to compare it with the previous results.

We are going to construct a precise theory behind the WEP attack in the subsequent chapters. All our analysis has been checked precisely through extensive amount of experiments. We show that we can recover a 104-bit long WEP key using 19 800 packets in less than a minute using an ordinary PC. With less number of packets, the attack will run for a longer period.

2.5.3 Previous Attacks on WPA

Regarding the security of WPA, in 2004, Moen, Raddum and Hole [MRH04] discovered that the recovery of at least two RC4 packet keys in WPA leads to a full recovery of the temporal key and the message integrity check key. When two keys are successfully recovered from the same segment

of 2^{16} consecutive packets, the Moen, Raddum and Hole attack can be applied. This leads to a TK key recovery attack on WPA with average complexity of 2^{104} using 2 packets. Almost all known and new key recovery attacks on WEP could have been applied to WPA if there were several packets using the same RC4 key. Indeed, only the Fluhrer, Mantin and Shamir attack [FMS01] is filtered. However, WPA uses a different secret key for every encrypted packet.

In 2009, Tews and Beck [BT09] found a practical attack on WPA-PSK to inject data in an encrypted channel. Note that this attack does not recover the encryption key and needs some additional quality of services features (described by IEEE 802.11e) which are not activated by default.

Algorithm 2.2 The phase2 of the WPA key hashing function.

phase2, step1:

1: $\mathsf{PPK}[0] = \mathsf{TTAK}[0]$
2: $\mathsf{PPK}[1] = \mathsf{TTAK}[1]$
3: $\mathsf{PPK}[2] = \mathsf{TTAK}[2]$
4: $\mathsf{PPK}[3] = \mathsf{TTAK}[3]$
5: $\mathsf{PPK}[4] = \mathsf{TTAK}[4]$
6: $\mathsf{PPK}[5] = \mathsf{TTAK}[4] + \mathsf{IV16}$

phase2, step2:

7: $\mathsf{PPK}[0] = \mathsf{PPK}[0] + \mathsf{S}[\mathsf{PPK}[5] + \mathsf{Mk16}(\mathsf{TK}[1], \mathsf{TK}[0])]$
8: $\mathsf{PPK}[1] = \mathsf{PPK}[1] + \mathsf{S}[\mathsf{PPK}[0] + \mathsf{Mk16}(\mathsf{TK}[3], \mathsf{TK}[2])]$
9: $\mathsf{PPK}[2] = \mathsf{PPK}[2] + \mathsf{S}[\mathsf{PPK}[1] + \mathsf{Mk16}(\mathsf{TK}[5], \mathsf{TK}[4])]$
10: $\mathsf{PPK}[3] = \mathsf{PPK}[3] + \mathsf{S}[\mathsf{PPK}[2] + \mathsf{Mk16}(\mathsf{TK}[7], \mathsf{TK}[6])]$
11: $\mathsf{PPK}[4] = \mathsf{PPK}[4] + \mathsf{S}[\mathsf{PPK}[3] + \mathsf{Mk16}(\mathsf{TK}[9], \mathsf{TK}[8])]$
12: $\mathsf{PPK}[5] = \mathsf{PPK}[5] + \mathsf{S}[\mathsf{PPK}[4] + \mathsf{Mk16}(\mathsf{TK}[11], \mathsf{TK}[10])]$

13: $\mathsf{PPK}[0] = \mathsf{PPK}[0] + \mathsf{RotR1}[\mathsf{PPK}[5] + \mathsf{Mk16}(\mathsf{TK}[13], \mathsf{TK}[12])]$
14: $\mathsf{PPK}[1] = \mathsf{PPK}[1] + \mathsf{RotR1}[\mathsf{PPK}[0] + \mathsf{Mk16}(\mathsf{TK}[15], \mathsf{TK}[14])]$
15: $\mathsf{PPK}[2] = \mathsf{PPK}[2] + \mathsf{RotR1}[\mathsf{PPK}[1]]$
16: $\mathsf{PPK}[3] = \mathsf{PPK}[3] + \mathsf{RotR1}[\mathsf{PPK}[2]]$
17: $\mathsf{PPK}[4] = \mathsf{PPK}[4] + \mathsf{RotR1}[\mathsf{PPK}[3]]$
18: $\mathsf{PPK}[5] = \mathsf{PPK}[5] + \mathsf{RotR1}[\mathsf{PPK}[4]]$

phase2, step3:

19: $\mathsf{RC4KEY}[0] = \mathsf{high8}(\mathsf{IV16})$
20: $\mathsf{RC4KEY}[1] = (\mathsf{high8}(\mathsf{IV16}) \mid \mathsf{0x20})\ \&\ \mathsf{0x7f}$
21: $\mathsf{RC4KEY}[2] = \mathsf{low8}(\mathsf{IV16})$
22: $\mathsf{RC4KEY}[3] = \mathsf{low8}((\mathsf{PPK}[5] \oplus (\mathsf{RotR1}(\mathsf{TK}[1] \| \mathsf{TK}[0]))))$
23: **for** $i = 0$ to 5 **do**
24: $\quad \mathsf{RC4KEY}[4 + (2 * i)] = \mathsf{low8}(\mathsf{PPK}[i])$
25: $\quad \mathsf{RC4KEY}[5 + (2 * i)] = \mathsf{high8}(\mathsf{PPK}[i])$
26: **end for**

Chapter 3

Classification of the RC4 Biases and Some Useful Lemmas

In this chapter, we introduce some notations and lemmas which are used later in the attacks against WEP and WPA. Then, we categorize the useful biases against WEP and WPA into two categories: conditional and unconditional ones. We will discuss later in Chapters 7 and 8 regarding why this categorization is necessary.

3.1 Notations and Some Useful Lemmas

Throughout this thesis, we denote $\bar{K}[i] \stackrel{\mathsf{def}}{=} K[0]+\cdots+K[i]$. We let z denote the keystream derived from K using RC4. In the applications we are concerned, the first bytes of a plaintext frame are often known (see Figure 2.3), as well as the IV, the first 3 bytes of K. That is, we assume that the adversary can use the keystream z and the IV in a known plaintext attack.

We let I_0 be a set of integers, which represents the key byte indices which are already known. We call an I_0-clue a value clue for all $\bar{K}$ bytes whose index are in I_0. To begin with, we have $I_0 = \{0, 1, 2\}$ and $\mathsf{clue} = \mathsf{IV}$.

Given a set of indices I_0 and an index i, we assume that we have a list $\mathsf{row}^{\mathsf{RC4}}_{i|I_0}$ of $d_{i|I_0}$ vectors $(\bar{f}_j, \bar{g}_j, p_j, q_j)$, $j = 1, \ldots, d_{i|I_0}$ with functions $\bar{f}_j$ and the corresponding predicates $\bar{g}_j$ such that

$$\mathsf{Pr}\left[\bar{K}[i] = \bar{f}_j(z, \mathsf{clue}) | \bar{g}_j(z, \mathsf{clue})\right] = p_j$$

for some probability $p_j \neq \frac{1}{N}$ and

$$\mathsf{Pr}\left[\bar{g}_j(z, \mathsf{clue})\right] = q_j$$

where q_j is called the *density* of the bias.

For simplicity, we assume that for some given i, z, and clue, all suggested $\bar{f}_j(z, \mathsf{clue})$ for j's such that $\bar{g}_j(z, \mathsf{clue})$ holds, are pairwise distinct. We further assume that the events $\bar{K}[i] = \bar{f}_j(z, \mathsf{clue})$ with different i's are independent. We will also assume that $\bar{f}_j$ and $\bar{g}_j$ are of the form $\bar{f}_j(z, \mathsf{clue}) = f_j(h(z, \mathsf{clue}))$ and $\bar{g}_j(z, \mathsf{clue}) = g_j(h(z, \mathsf{clue}))$, where $\mu = h(z, \mathsf{clue})$ lies in a domain of size N_μ. In fact, h is just a function compressing the data to the minimum necessary to compute $\bar{f}_j$ and $\bar{g}_j$.

Definition 3.1. *Let A, B and C be three random variables over $\mathbf{Z}_N$. We say that A is biased towards B with bias p conditioned on an event E and we represent it as $A \overset{p}{\underset{E}{=}} B$ if*

$$\mathsf{Pr}(A - B = x|E) = \begin{cases} p & \text{if} \quad x = 0 \\ \\ \frac{1-p}{N-1} & \text{otherwise} \end{cases}$$

When $\mathsf{Pr}[E] = 1$, it is denoted as $A \overset{p}{=} B$.

Lemma 3.1. *Let A, B and C be random variables in $\mathbf{Z}_N$ such that*

$$A \overset{p_1}{=} B \qquad B \overset{p_2}{=} C$$

then we assume that $A - B$ and $B - C$ are independent. We have $A \overset{P}{=} C$, where

$$P = \frac{1}{N} + \left(\frac{N}{N-1}\right)\left(p_1 - \frac{1}{N}\right)\left(p_2 - \frac{1}{N}\right) \overset{\mathsf{def}}{=} p_1 \otimes p_2$$

Proof. For $x \neq 0$, we have

$$\begin{aligned} \mathsf{Pr}[C - A = x] &= \sum_{y} \mathsf{Pr}[B - A = y] \;.\; \mathsf{Pr}[C - B = x - y] \\ &= \sum_{\substack{y \neq 0 \\ y \neq x}} \mathsf{Pr}[B - A = y] \;.\; \mathsf{Pr}[C - B = x - y] \\ &\quad + \mathsf{Pr}[A = B] \;.\; \mathsf{Pr}[C - B = x] + \mathsf{Pr}[B - A = x] \;.\; \mathsf{Pr}[B = C] \\ &= (N-2)\left(\frac{1-p_1}{N-1}\right)\left(\frac{1-p_2}{N-1}\right) + p_1\left(\frac{1-p_2}{N-1}\right) + p_2\left(\frac{1-p_1}{N-1}\right) \end{aligned}$$

which does not depend on x. Then,

$$\mathsf{Pr}[A = C] = 1 - \sum_{x \neq 0} \mathsf{Pr}[C - A = x] = \frac{1}{N} + \left(\frac{N}{N-1}\right)\left(p_1 - \frac{1}{N}\right)\left(p_2 - \frac{1}{N}\right)$$

So, $A \overset{P}{=} C$. □

Corollary 3.1. *Let A, B, C, D and E be random variables in $\mathbf{Z}_N$ such that*

$$A \overset{p_1}{=} B \qquad B \overset{p_2}{=} C \qquad C \overset{p_3}{=} D \qquad D \overset{p_4}{=} E$$

then we assume that $A - B$, $B - C$, $C - D$ and $D - E$ are independent. We have $A \overset{P}{=} E$, where

$$P = p_1 \otimes p_2 \otimes p_3 \otimes p_4 = \frac{1}{N} + \left(\frac{N}{N-1}\right)^3 \cdot \prod_{i=1}^{4}\left(p_i - \frac{1}{N}\right)$$

For $p_4 = 1$, we obtain

$$P = p_1 \otimes p_2 \otimes p_3 = \frac{1}{N} + \left(\frac{N}{N-1}\right)^2 \cdot \prod_{i=1}^{3}\left(p_i - \frac{1}{N}\right)$$

Proof. The $\otimes$ operation is commutative and associative over $[0,1]$ and 1 is the neutral element. The above statements should be trivial using these properties. □

We can extend the above Corollary by adding new conditions.

Lemma 3.2. *Let A, B, C, D and E be random variables in $\mathbf{Z}_N$ and Cond and Cond' be two events such that*

$$A \overset{p_1}{=} B \qquad B \overset{p_2}{=} C \qquad C \underset{\mathsf{Cond}'}{\overset{p_3}{=}} S[D] \qquad D \overset{p_4}{=} E$$

We assume that $A - B$, $B - C$, $C - S[D]$, $D - E$ and Cond' are independent; Furthermore, we assume

$$(A = S[D] \wedge \mathsf{Cond}) \Leftrightarrow (A = S[D] \wedge \mathsf{Cond}') \qquad \textit{and} \qquad \mathsf{Pr}[\mathsf{Cond}] = \mathsf{Pr}[\mathsf{Cond}']$$

Assuming that

$$\mathsf{Pr}[A = S[E] | A \neq S[D], D \neq E, \mathsf{Cond}] = \frac{1}{N-1}$$

we have

$$\mathsf{Pr}[A = S[E] | \mathsf{Cond}] = p_1 \otimes p_2 \otimes p_3 \otimes p_4$$

Proof.

$$\begin{aligned} \mathsf{Pr}[A = S[D] = S[E] | \mathsf{Cond}] &= \mathsf{Pr}[A = S[D] = S[E] | \mathsf{Cond}'] \left(\tfrac{\mathsf{Pr}[\mathsf{Cond}']}{\mathsf{Pr}[\mathsf{Cond}]} \right) \\ &= \mathsf{Pr}[A = S[D], D = E | \mathsf{Cond}'] \\ &= \mathsf{Pr}[A = S[D] | \mathsf{Cond}'] \ . \ \mathsf{Pr}[D = E] \\ &= (p_1 \otimes p_2 \otimes p_3) \ . \ p_4 \end{aligned}$$

With the extra assumption

$$\mathsf{Pr}[A = S[E] | A \neq S[D], D \neq E, \mathsf{Cond}] = \frac{1}{N-1}$$

we obtain

$$\mathsf{Pr}[A = S[E] | \mathsf{Cond}] = p_1 \otimes p_2 \otimes p_3 \otimes p_4$$

□

We use a critical relation between the key bytes, j and the bytes of the state, which we summarize it as the following lemma.

Lemma 3.3. *To avoid the key byte dependency, the following equation can be extracted to have a better key recovery attack.*

$$\bar{K}[i] = j_i - \sum_{x=1}^{i} S_{x-1}[x]$$

Proof. We prove it by induction on i by using

$$j_i = j_{i-1} + S_{i-1}[i] + K[i]$$

□

Lemma 3.4. *For $0 \leq t < i$, the following five relations hold on RC4$^\star(t)$ for any set $(m_1, \ldots, m_b)$ of pairwise different m_j's such that $m_j \leq t$ or $m_j > i-1$.*

$$P_A^b(i,t) \stackrel{\mathsf{def}}{=} \mathsf{Pr}\left[\bigwedge_{j=1}^{b} S_{i-1}[m_j] = \cdots = S_{t+1}[m_j] = S_t[m_j]\right] = \left(\tfrac{N-b}{N}\right)^{i-t-1}$$

$$S_{i-1}[m_j] \stackrel{P_A^1}{=} S_t[m_j]$$

$$\sum_{x=1}^{i} S_{x-1}[x] \stackrel{P_B(i,t)}{=} \sigma_i(t) \quad \textit{with} \quad P_B(i,t) \stackrel{\mathsf{def}}{=} \prod_{k=0}^{i-t-1}\left(\frac{N-k}{N}\right) + \frac{1}{N}\left(1 - \prod_{k=0}^{i-t-1}\left(\frac{N-k}{N}\right)\right)$$

$$P_0 \stackrel{\mathsf{def}}{=} \mathsf{Pr}[S'_{i-1}[i] = \cdots = S'_1[i] = S_{N-1}[i] = \cdots = S_i[i]] = \left(\tfrac{N-1}{N}\right)^{N-2}$$

$$S'_{i-1}[i] \stackrel{P_0}{=} S_i[i]$$

where m_j's are distinct and

$$\sigma_i(t) = \sum_{j=0}^{t} S_{j-1}[j] + \sum_{j=t+1}^{i} S_t[j]$$

Proof. Note that $S_{i-1}[m_j] = S_t[m_j]$ is equivalent to $S_{i-1}[m_j] = \cdots = S_{t+1}[m_j] = S_t[m_j]$. Furthermore, $P_A^b(i,t)$ is defined as the probability that a set of bytes corresponding to a set of indexes $(m_1, \ldots, m_b)$ are not swapped from S_t to S_{i-1}. Since $m_j \leq t$ or $m_j > i-1$, this set of indices will not be selected by the index i from S_t to S_{i-1}. Hence, they can only be picked by the index j which moves uniformly at random by the definition of RC4$^\star(t)$. So, this is correct with probability $\left(\frac{N-b}{N}\right)^{i-t-1}$. In fact, we have

$$S_{i-1}[m_j] \stackrel{P_A^1}{=} S_t[m_j]$$

That is because

$$\underset{x \neq y}{\mathsf{Pr}}\left[S_{i-1}[m_j] = y | S_t[m_j] = x\right] = \frac{1}{N-1}$$

Since we know up to state S_t, we have to approximate $\sum_{x=1}^{i} S_{x-1}[x]$ with the state bytes in S_t. The first term in $P_B(i,t)$ is the probability that $S_{x-1}[x]$ can be approximated as $S_t[x]$ for $x > t+1$. The second term is the probability that at least one of these approximations is wrong, but at the end the result holds with uniform probability. We can also assume that

$$\underset{y \neq \sigma_i(t)}{\mathsf{Pr}}\left[\sum_{x=1}^{i} S_{x-1}[x] = y\right] = \frac{1}{N-1}$$

P_0 is the probability that the index i is not swapped from S_i to S'_{i-1}. This probability depends only on the values of j and j', which change uniformly at random in RC4$^\star(t)$. There are $N-2$ state updates in the way, so the overall probability is $\left(\frac{N-1}{N}\right)^{N-2}$. We also have

$$\underset{x \neq y}{\mathsf{Pr}}\left[S'_{i-1}[i] = y | S_i[i] = x\right] = \frac{1}{N-1}$$

This leads to $S'_{i-1}[i] \stackrel{P_0}{=} S_i[i]$. □

3.2 Classification of the Biases

In this section, we classify the biases on RC4. We only report those which are exploitable against WEP and WPA. All such biases would be elaborated in the next chapter and their success probability would be extracted in our model. The list of biases includes the improved version of the Klein attack in [VV07], the improved version of the Maitra-Paul attack in [MP08] and two other attacks in [SVV10]. Furthermore, it includes an improved version of 19 biases by Korek [Kor04b, Kor04a] and SVV_10, the improved bias of Sepehrdad, Vaudenay and Vuagnoux in [SVV10]. All the probabilities are new.

We finally classify the biases (those exploitable against WEP and WPA) into two categories: conditional and unconditional ones. We use these notions specifically in the WPA attack section later. Although all the biases are conditional, we put the SVV_10 and the Korek biases in the conditional category and the Klein Improved, Maitra-Paul, SVV_008 and SVV_009 biases in the unconditional category. This is because the conditions of the unconditional biases hold with a high probability. This is not the case for conditional biases.

Chapter 4

The Biases in the KSA and the PRGA of RC4

In this chapter, we are going to recall the state of the art correlations in RC4, exploitable against WEP and WPA. We explore three strategies: biases in the KSA, biases in the PRGA and the black box analysis. Then, we propose an exhaustive strategy to go empirically through all the linear correlations in the PRGA of RC4 and find the significant ones. Later, some of those correlations were proved in [SGMPS11a]. Exhausting over all such correlations are very expensive. Hence, to drop the complexity, we will introduce a technique based on Fourier transform analysis. Often, biases we derive using this strategy are not exploitable against WEP and WPA, but they are all valid for RC4 and might be used later by the motivated researchers to launch some other attacks against RC4.

4.1 Known Correlations in the KSA of RC4

4.1.1 Roos Correlation

In September 1995, Andrew Roos [Roo95] introduced a strong bias between the output bytes of the KSA and the bytes of the key. This bias exists up to the 48-th byte. After this byte, the probabilities are extremely close to the uniform distribution. He computed this bias empirically, but then this correlation was proved in [PM07]. More precisely,

Lemma 4.1 (**The Roos Lemma**). *The most likely value for $S_{N-1}[i]$ at the end of the* KSA *is*

$$S_{N-1}[i] \overset{P_{\mathsf{Roos}}(i)}{=} \bar{K}[i] + \frac{i(i+1)}{2}$$

where

$$P_{\mathsf{Roos}}(i) = P_B(i,-1)\left(\frac{N-i}{N}\right)\left(\frac{N-1}{N}\right)^{N-1} + \frac{1}{N}\left(1 - P_B(i,-1)\left(\frac{N-i}{N}\right)\left(\frac{N-1}{N}\right)^{N-1}\right)$$

Proof. We already know from Lemma 3.4 that

$$\bar{K}[i] \overset{P_B(i,-1)}{=} j_i - \sigma_i(-1)$$

We can also compute

$$\mathsf{Pr}[S_{N-1}[i] = S_{-1}[j_i] = j_i] = \left(\frac{N-i}{N}\right)\left(\frac{N-1}{N}\right)^{N-1}$$

By the above two relations, we have

$$\begin{aligned}\mathsf{Pr}\left[S_{N-1}[i] = \bar{K}[i] + \tfrac{i(i+1)}{2}\right] &= P_B(i,-1)\left(\tfrac{N-i}{N}\right)\left(\tfrac{N-1}{N}\right)^{N-1} \\ &\quad + \tfrac{1}{N}\left(1 - P_B(i,-1)\left(\tfrac{N-i}{N}\right)\left(\tfrac{N-1}{N}\right)^{N-1}\right)\end{aligned}$$

□

4.1.2 Mantin Correlation in the Output Bytes of the KSA

In his master thesis, Itsik Mantin [Man01] proved that by picking a random key, the output bytes of the KSA are not uniformly distributed. However, in the literature of the analysis of RC4 and specifically the PRGA algorithm, it is often assumed that the output of the KSA is a random permutation chosen from the set of all permutations over $\{0, \ldots, N-1\}$ due to the very small bias of the output of the KSA.

Lemma 4.2 (Corollary 6.3.1 in [Man01]). *At the end of the* KSA*, for* $0 \leq x < N$ *and* $0 \leq y < N$*, we have*

$$\mathsf{Pr}[S_{N-1}[x] = y] = \begin{cases} \frac{1}{N}\left[\left(\frac{N-1}{N}\right)^y + \left(1 - \left(\frac{N-1}{N}\right)^y\right)\left(\frac{N-1}{N}\right)^{N-x-1}\right] & \text{if} \quad y \leq x \\ \frac{1}{N}\left[\left(\frac{N-1}{N}\right)^{N-x-1} + \left(\frac{N-1}{N}\right)^y\right] & \text{if} \quad y > x \end{cases}$$

4.2 Known Correlations in the PRGA of RC4

4.2.1 Jenkins Correlation (The Glimpse Property)

In 1996, Robert Jenkins described two biased correlations experimentally found on the PRGA of RC4 on his website [Jen96]. The first correlation considers the case where $S'_i[i] + S'_i[j'_i] = j'_i$. Thus, the i-th keystream byte is given by

$$z_i = S'_i[S'_i[i] + S'_i[j'_i]] = S'_i[j'_i] = j'_i - S'_i[i]$$

which holds with probability $2/N$ instead of $1/N$ with $i = 1, 2, \ldots$. The second correlation appears when $S'_i[i] + S'_i[j'_i] = i$. In this case, the i-th byte of the keystream is given by

$$z_i = S'_i[S'_i[i] + S'_i[j'_i]] = S'_i[i] = i - S'_i[j'_i]$$

which holds with probability $2/N$. More precisely,

Theorem 4.1 (Jenkins correlation [Jen96], Sec. 2.3 in [Man01]). *Assume that the initial permutation* $S'_0 = S_{N-1}$ *is randomly chosen from the set of all the possible permutations over* $\{0, \ldots, N-1\}$*. Then,*

$$\mathsf{Pr}[S'_i[j'_i] = i - z_i] \approx \tfrac{2}{N} \qquad \mathsf{Pr}[S'_i[i] = j'_i - z_i] \approx \tfrac{2}{N}$$

Proof.

$$\begin{aligned}\mathsf{Pr}[S'_i[j'_i] = i - z_i] &= \mathsf{Pr}[S'_i[j'_i] = i - z_i | S'_i[i] + S'_i[j'_i] = i] \;.\; \mathsf{Pr}[S'_i[i] + S'_i[j'_i] = i] \\ &\quad + \mathsf{Pr}[S'_i[j'_i] = i - z_i | S'_i[i] + S'_i[j'_i] \neq i] \;.\; \mathsf{Pr}[S'_i[i] + S'_i[j'_i] \neq i] \\ &= \tfrac{1}{N} + \tfrac{1}{N}\left(1 - \tfrac{1}{N}\right) \approx \tfrac{2}{N}\end{aligned}$$

By symmetry, the other equation can be proved similarly. □

We use this bias later to derive the Klein correlation in order to mount a key recovery attack on RC4. Mantin generalized the Glimpse property in [Man01], where different relations between i and z_i hint on the corresponding relations between $S'_i[i]$ and $S'_i[j'_i]$. It is given as the following theorem:

Theorem 4.2 (Sec. 7 of [Man01]). *Let $f : \mathbf{Z}_N \to \mathbf{Z}_N$ and let $h_f(x) \stackrel{\mathsf{def}}{=} f(x) + x$ and h_f is one-to-one from $\mathbf{Z}_N$ to $\mathbf{Z}_N$. Then, for every $i \in \mathbf{Z}_N$, we have*

$$\mathsf{Pr}[S'_i[j'_i] = f(S'_i[i]) | h_f(Z(S'_i, i, j'_i)) = i] \approx \frac{2}{N}$$

where $Z(S'_i, i, j'_i) = S'_i[S'_i[i] + S'_i[j'_i]]$.

The original glimpse is a special case with $f(x) \stackrel{\mathsf{def}}{=} i - z_i$ and $h_f(z) = i$.

Later, this theorem was extended in [Man05a] and Mantin discussed the availability of the glimpse property in many other output selection functions. He proposed a conjecture to generalize the Jenkins correlation and claimed that the glimpse property exists for almost any output selection function of depth two.

Conjecture 4.1 (The Generalized Glimpse Conjecture). *Let $f, g : \mathbf{Z}_N \to \mathbf{Z}_N$ be invertible functions and denote their inverse functions by F and G respectively. Let $h : \mathbf{Z}_N \times \mathbf{Z}_N \to \mathbf{Z}_N$ be a 2-parameter function that is invertible in each of its parameters and let H_1 and H_2 be the inverse functions of h, where $\forall X, Y \in \mathbf{Z}_N$, we have*

$$H_1(X, h(X, Y))) = Y \qquad H_2(h(X, Y), Y) = X$$

Let $Z(S'_i, i, j'_i) \stackrel{\mathsf{def}}{=} f(S[h(S[g(i)], S[j])])$ be an output selection function of an RC4-like keystream generator. then, we have

$$\mathsf{Pr}[S'_i[j'_i]] = H_1(F(Z(S'_i, i, j'_i)), g(i))] \approx \frac{2}{N}$$

For the intuition behind this conjecture, refer to [Man05a].

4.2.2 Mantin and Shamir Correlation

In 2001, Mantin and Shamir [MS01] discovered a huge bias for the second keystream word z_2 of RC4. Currently, this bias is the largest bias existing in keystreams of RC4. They constructed a strong distinguisher for RC4, which requires only N output words. Using this bias, the second byte of the plaintext can be recovered if RC4 is used in a broadcast encryption scheme. In such schemes, their attack was extended in [SPSG11] to recover bytes 3 to 255 of the plaintext using N^3 ciphertexts by providing a bias for z_r towards zero when $2 < r < 256$.

Lemma 4.3 (Theorem 1 in [MS01]). *Assume that the initial permutation $S'_0 = S_{N-1}$ is randomly chosen from the set of all the possible permutations over $\{0, \ldots, N-1\}$. Then, the probability that the second output word of RC4 is 0 is approximately $2/N$. In fact, we have $z_2 \overset{\frac{2}{N}}{=} 0$.*

Proof. First, we show that if $S_{N-1}[2] = 0$ and $S_{N-1}[1] \neq 2$, we obtain $z_2 = 0$. Assume that $S'_0[1] = \alpha$ and $S'_0[\alpha] = \beta$, then $i = 1$ and $j'_1 = S'_0[1] = \alpha$, then we swap $S'_0[1]$ and $S'_0[\alpha]$. In the next iteration, $i = 2$ and $j'_2 = \alpha + S'_1[2] = \alpha$, that is because we assumed $S_{N-1}[1] \neq 2$ and $S_{N-1}[2] = 0$, so $S'_1[2] = 0$, then we swap $S'_1[2]$ and $S'_1[\alpha]$ and z_2 is computed as $z_2 = S'_2[S'_2[2]+S'_2[\alpha]] = S'_2[\alpha] = 0$. Finally,

$$\begin{aligned}
\mathsf{Pr}[z_2 = 0] &= \mathsf{Pr}[z_2 = 0 | S'_0[2] = 0, S'_0[1] \neq 2] \;.\; \mathsf{Pr}[S'_0[2] = 0, S'_0[1] \neq 2] \\
&+ \mathsf{Pr}[z_2 = 0 | S'_0[2] \neq 0 \vee S'_0[1] = 2] \;.\; \mathsf{Pr}[S'_0[2] \neq 0 \vee S'_0[1] = 2] \\
&= \frac{1}{N}\left(\frac{N-1}{N}\right) + \frac{1}{N}\left[\left(\frac{N-1}{N}\right) + \frac{1}{N} - \frac{1}{N}\left(\frac{N-1}{N}\right)\right] \\
&= \frac{1}{N}\left(\frac{N-1}{N}\right)\left(2 - \frac{1}{N}\right) + \frac{1}{N^2} \\
&\approx \frac{2}{N}
\end{aligned}$$

If $x \neq 0$, we also have

$$\mathsf{Pr}[z_2 = x] = \frac{1 - \mathsf{Pr}[z_2 = 0]}{N-1} = \frac{N-2}{N(N-1)}$$

□

4.2.3 Paul, Rathi and Maitra Correlation

In 2008, Paul, Rathi and Maitra [PRM08] described a biased correlation on the index of first keystream byte of RC4. Using this correlation, they found a bias between the three first words of the secret key and the first keystream word z_1 of RC4. We recall the first Lemma here and describe the second one in Section 4.3.

Lemma 4.4. *Assume that the initial state S'_0 is chosen uniformly at random from the set of all possible permutations of the set $\{0, 1, \ldots, N-1\}$. Then the probability distribution of the output index $S'_1[1] + S'_1[S'_0[1]] = S'^{-1}_1[z_1]$ that selects the first byte of the keystream output is given by*

$$\mathsf{Pr}\left(S'_1[1] + S'_1[j'_1] = x\right) = \begin{cases} \frac{1}{N} & \text{for odd } x \\ \frac{1}{N} - \frac{2}{N(N-1)} & \text{for even } x \neq 2 \\ \frac{2}{N} - \frac{1}{N(N-1)} & \text{for even } x = 2 \end{cases}$$

4.2.4 Mironov Correlation

In 2002, Ilya Mironov [Mir02] observed a "sine-curve-like" probability distribution of the first keystream byte of RC4. This strange probability distribution was proved recently in [SGMPS11b]. Accordingly, they also proved that the first keystream byte of RC4 is negatively biases towards zero. We only recall the negative bias of z_1 towards zero here. The full distribution function of z_1 is given in [SGMPS11b]. The sinusoidal curve from [SGMPS11b] is depicted in Figure 4.1.

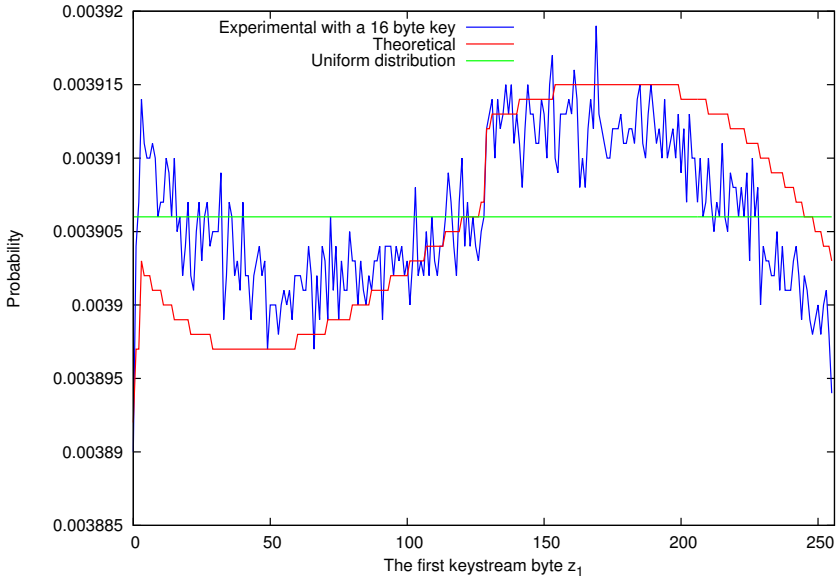


Figure 4.1: probability distribution of z_1 (both in theory and practice)

Lemma 4.5. *Assume that the initial permutation S_{N-1} of the* PRGA *of RC4 is randomly chosen from the set of all permutations of $\{0, 1, \ldots, N-1\}$. Then the probability that the first output byte of RC4 keystream z_1 is 0 is approximately $1/N - 1/N^2$.*

Mironov [Mir02] also proposed to discard the first 512 keystream bytes of RC4 to avoid the initial weaknesses in the keystream. Indeed, none of the key recovery attacks on RC4 is applicable if 512 bytes of the keystream are discarded.

4.3 Binding the State of the Art KSA and PRGA Weaknesses

None of the weaknesses in the KSA and the PRGA can be exploited for RC4 if they can not be bound together. This means that the attacker should find a way to relate the biases existing for the KSA and the PRGA of RC4. One of the first of such efforts was done by Andreas Klein, who merged the weaknesses of the KSA and Jenkins bias for the PRGA. In this section, we are going to describe how this binding was performed. We later describe more new bindings in Chapters 5 and 6 which did not exist in the literature before. In the same chapter, we show that the Klein bias and many more biases can still be improved by recovering the sum of the key bytes instead of the key bytes individually.

4.3.1 Klein Correlation

In 2006, Andreas Klein demonstrated a binding between the KSA weaknesses and Jenkins correlation (see Theorem 4.1) in the PRGA. From Lemma 3.4 and the step 6 of the KSA, we derive

1. $S'_i[j'_i] \overset{P_J}{=} i - z_i$ (Lemma 4.1)

2. $S'_i[j'_i] = S'_{i-1}[i]$
3. $S'_{i-1}[i] \overset{P_0}{=} S_i[i]$ (Lemma 3.4)
4. $S_i[i] = S_{i-1}[j_i]$
5. $j_i = S_{i-1}[i] + j_{i-1} + K[i]$

By merging all the above relations and using Lemma 3.1, we obtain

$$K[i] \overset{P_K(i)}{=} S_{i-1}^{-1}[i - z_i] - S_{i-1}[i] - j_{i-1}$$

where

$$P_K(i) = P_J P_0 + \frac{1}{N-1}(1 - P_J)(1 - P_0) = P_J \otimes P_0$$

The attacker recursively recovers j_i's and $\bar{K}[i]$'s. If any $\bar{K}[i]$ is recovered incorrectly, all the key bytes after will be recovered incorrectly as well.

4.3.2 Paul, Rathi and Maitra Correlation

In 2008, Paul, Rathi and Maitra [PRM08] derived a correlation between the first three bytes of the key and the first keystream byte of RC4 when $N = 256$ and $\ell = 16$. This correlation was derived by merging the Roos correlation and Lemma 4.4 (see [PRM08] for proof). We will use this correlation later in Sec. 6.2.7.

Lemma 4.6. *For any arbitrary secret key, the correlation between the key bytes and the first byte of the keystream output is given by*

$$\mathsf{Pr}[z_1 = \bar{K}[2] + 3] = \xi = \frac{1}{N}\left[\left(\frac{N-1}{N}\right)^N \left(1 - \frac{1}{N} + \frac{1}{N^2}\right) + \frac{1}{N^2} + 1\right]$$

This bias can not be extended to other bytes of the key as it depends on the index of the first keystream word.

4.4 Exhausting over All Linear Correlations in the PRGA

In general, the methods used to find correlations in RC4 are either opportunistic or not given. Papers tend to describe the characteristics of the biases without revealing the techniques used to discover them. We propose to describe some simple but efficient techniques to highlight weaknesses in the PRGA through exhaustive search on a subset of elements.

We define a set of linear equations which contains all the known biased correlations of the PRGA described in the previous section.

Our objective is to highlight linear correlations between the internal values of a round of the PRGA and the keystream word generated by this round; that is, the subset of elements $\{i, j'_i, S'_i[i], S'_i[j'_i], z_i\}$. The correlations previously discovered by Jenkins, Mantin and Shamir and

Paul, Rathi and Maitra must be rediscovered with this method. Surprisingly, some new biases are found as well. Later in the attack on WEP or WPA, we do not use any of these biases, as we could not merge them with the KSA weaknesses.

We define the linear equations as

$$a_0 \cdot i + a_1 \cdot j'_i + a_2 \cdot S'_i[i] + a_3 \cdot S'_i[j'_i] + a_4 \cdot z_i \quad = \quad b \tag{4.1}$$

where the a_i's are elements of $\mathbf{Z}/256\mathbf{Z}$ and b is a fixed value in $\mathbf{Z}/256\mathbf{Z}$. This defines 2^{48} linear equations.

To reduce this number, we decompose these equations into 256 subgroups. Each of them corresponds to a specific round (i.e., i is fixed). Thus, both $a_0 \cdot i$ and b can be merged into one value and Eq. (4.1) becomes

$$c_0 \cdot j'_i + c_1 \cdot S'_i[i] + c_2 \cdot S'_i[j'_i] + c_3 \cdot z_i \quad = \quad C \tag{4.2}$$

where $C = (b - a_0 \cdot i)$ and c_i's are elements of $\mathbf{Z}/256\mathbf{Z}$. Since the number of linear equations is still too large, we limit the coefficients set of the c_i's to $\{-1, 0, 1\}$. Indeed, this set is enough to include all the previously known biased correlation in the PRGA. We obtain 256 graphs of 81 linear equations.

We compute the 256 first rounds of the PRGA with 10^9 randomly chosen RC4 secret keys of 16 bytes and we verify all the linear equations described by Eq. (4.2). For every equation, a counter is incremented when it holds.

Below, we give the biased correlations found for the 256 first keystream bytes (from z_1 to z_{256}) generated by the PRGA. Every coefficient c_i has been replaced by the corresponding element to provide an easier reading of the table (i.e., j'_i must be read as c_0, $S'_i[i]$ as c_1, etc). Correlations with z_i (i.e., $c_3 \neq 0$) are called New_XXX and biases without z_i (i.e., $c_3 = 0$) are named New_noz_XXX.

In Figure 4.2, we confirm the presence of known biases such as the Jenkins correlations. More interestingly, new biases in the PRGA appear. Some of them have a probability of success which depends on the value of i.

Some rounds of the PRGA provide additional biased correlations. In Figure 4.3, we give extra biases which appear in round 1. Figure 4.4 depicts the additional correlations in the second round of the PRGA. Finally, Figure 4.5 describes further biased correlations in rounds 0 mod 16 of the PRGA. In Chapter 5, we bind New_008 and New_009 biases with the KSA weaknesses.

The probability of the biased correlations New_001 and New_002 depends on the round of the PRGA and the value C. In Figure 4.6, we show the success probability of New_001 according to C and the rounds 1, 16, 32, 64, 128, 192 and 256 of the PRGA. Similarly, we compute the evolution of the success probability of the biased correlations New_002 in Figure 4.7.

4.4.1 Spectral Approach to Derive New Biases

The brute force approach of the previous section is too expensive. If we do not restrict the coefficients to be in the set $\{-1, 0, 1\}$, it is infeasible to find the best biases using the exhaustive search approach. In the following, we propose a technique to find all such correlations in a reasonable amount of time using the Fast Fourier Transform (FFT) algorithm. First, we define the Fourier Transform of a function. Unless otherwise mentioned, $\mathbf{G}$ represents the group $\mathbf{Z}_p$.

j'_i	$S'_i[i]$	$S'_i[j'_i]$	z_i	C	Prob ($i=3$)	Remark
1	-1	0	-1	0	$2/N$	Jenkins
0	0	1	1	i	$2/N$	Jenkins
0	1	1	-1	0	$1.9/N$	New_000
0	1	1	-1	1	$0.89/N$	New_001
⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	1	-1	255	$1.25/N$	New_001
0	1	1	1	0	$0.95/N$	New_002
⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	1	1	255	$0.95/N$	New_002
1	1	0	0	0	$0.95/N$	New_noz_000
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	0	0	255	$0.95/N$	New_noz_000
1	1	-1	0	i	$2/N$	New_noz_001
1	-1	1	0	i	$2/N$	New_noz_002
1	-1	0	0	1	$0.9/N$	New_noz_003
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	-1	0	0	255	$1.25/N$	New_noz_003
1	-1	0	0	0	$1.9/N$	New_noz_004
0	0	1	0	$i+1$	$1.36/N$	New_noz_005
⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	1	0	255	$0.9/N$	New_noz_005
0	0	1	0	i	$2.34/N$	New_noz_006

Figure 4.2: The correlations experimentally observed for rounds $1, 2, 3, \ldots, 256$ of the PRGA. Note that probability of the biases New_000, New_noz_005 and New_noz_006 decreases according to i. The probabilities given in this table correspond to round 3 (i.e., $i = 3$). New_noz_004 and New_noz_006 are not biased when $i = 1$.

Definition 4.1. *Let p be an integer and $\theta = e^{\frac{2i\pi}{p}}$. The Discrete Fourier Transform (DFT) of a function f over $\mathbf{G}^s$ is defined as*

$$\hat{f}(c) = \sum_{x \in \mathbf{G}^s} f(x)\theta^{-c \bullet x}$$

where $\bullet$ is the dot product.

We compute the Fourier Transform of the type f of the distribution that some state bytes of RC4 are following. Deploying this method over $\mathbf{Z}_{256}^s$, we can derive some *good* linear relations. We call a linear relation good if the probability of Eq. (4.1) occurring is much higher than expected ($\gg \frac{1}{N}$). We use the 4-tuple defined above as an example. In fact, we can use exactly the same method to derive the biases for the linear relations in Sec. 4.5.2 between the secret key and the keystream. To deal with this problem, we assume $\mathbf{G} = \mathbf{Z}_{256}$ and we fix i. Then, we query RC4 for $\mathcal{N}$ vectors $\mathcal{V}_t \in (j'_i, S'_i[i], S'_i[j'_i], z_i)_t \in \mathbf{G}^4$ corresponding to $\mathcal{N}$ distinct keys, where $1 \leq t \leq \mathcal{N}$.

j_1'	$S_1'[1]$	$S_1'[j_1']$	z_1	C	Prob	Remark
0	1	0	-1	0	$0.95/N$	New_003
0	1	1	0	2	$1.95/N$	Paul, Rathi, Maitra
1	1	0	0	2	$1.94/N$	New_noz_014

Figure 4.3: Additional biased correlations experimentally observed using Eq. (4.2) in the first round of the PRGA (i.e., $i = 1$).

j_2'	$S_2'[2]$	$S_2'[j_2']$	z_2	C	Prob	Remark
0	0	0	1	0	$2/N$	Mantin and Shamir
1	-1	1	-1	0	$2/N$	New_004
1	1	0	-1	even	$1.0183/N$	New_005
1	1	0	-1	odd	$1.0316/N$	New_006
1	0	1	0	6	$2.37/N$	New_noz_007
1	0	-1	0	255	$0.75/N$	New_noz_008
1	-1	1	0	0	$2/N$	New_noz_009
0	-1	1	0	0	$0.95/N$	New_noz_010

Figure 4.4: Additional biased correlations experimentally observed using Eq. (4.2) in the second round of the PRGA ($i = 2$). Note that the probability of successes of correlations New_005 and New_006 decrease according to the value C.

Finally, we define a function f as follows:

$$f(x) = \text{counter}(x) = \frac{1}{\mathcal{N}} \sum_{t=1}^{\mathcal{N}} \mathbf{1}_{\mathcal{V}_t = x}$$

where $x \in \mathbf{G}^4$ and $\mathbf{1}_{\mathcal{V}_t=x}$ is a random variable which is 1 if $\mathcal{V}_t = x$ and it is zero otherwise. In fact, f is the *type* of the distribution the 4-tuples are following. We query RC4 and compute the the numerical values of the f function and store it in a table. Deploying the Discrete Fourier Transform (DFT) on f, we have

$$\hat{f}(c) = \sum_{x} \theta^{-c \,\bullet\, x} f(x) = \sum_{C} \sum_{(x \,:\, c \,\bullet\, x = C)} \theta^{-C} f(x) = \sum_{C} \theta^{-C} \cdot \Pr[c \,\bullet\, \mathcal{V} = C] \tag{4.3}$$

for a random vector $\mathcal{V}$, where $\bullet$ is the dot product. Then, we follow this approach: we compute $|\hat{f}(c)|^2$ for c's such that this norm is high. Filtering those c's yields "good" c's. More clearly, we construct the table $|\hat{f}(c)|^2$ of all linear masks and filter out c's that lead to small value for $|\hat{f}(c)|^2$. We end up with some almost-good c's. Then, we can exhaustively search in the list of almost-good c's left and find the good ones.

The above technique works because in the last sum in Eq. (4.3) we are working in a complex circle. Assuming that the probability of Eq. (4.2) for all incorrect c's to be uniform and to be high for the the correct c, all the complex vectors in the circle would cancel out each other except the one that corresponds to the correct c. Therefore, using this method one can find a list of almost-good c's. Then, one would run an exhaustive search over C and check all almost-good c's. We only keep those that yield a probability much higher than $\frac{1}{N}$ for Eq. (4.2).

This method is much faster than exhaustive search. Since we currently do not know how to merge biases of the PRGA when $S_i'[i]$ is involved, we only consider the linear relation between the

j'_i	$S'_i[i]$	$S'_i[j'_i]$	z_i	C	Prob	Remark
0	0	0	1	-i	$1.0411/N$	New_007
0	0	1	-1	i	$1.0500/N$	New_008
0	0	1	1	-i	$1.0338/N$	New_009
0	1	1	0	-i	$1.1107/N$	New_noz_011
0	1	0	0	-i	$1.1276/N$	New_noz_012
0	1	-1	0	-i	$1.1067/N$	New_noz_013

Figure 4.5: Additional biased correlations experimentally observed using Eq. (4.2) in rounds 0 mod 16 of the PRGA. Note that the probability of success of these correlations decreases according to the value i and become barely exploitable when $i > 48$. Probabilities given in this table come from the round 16.

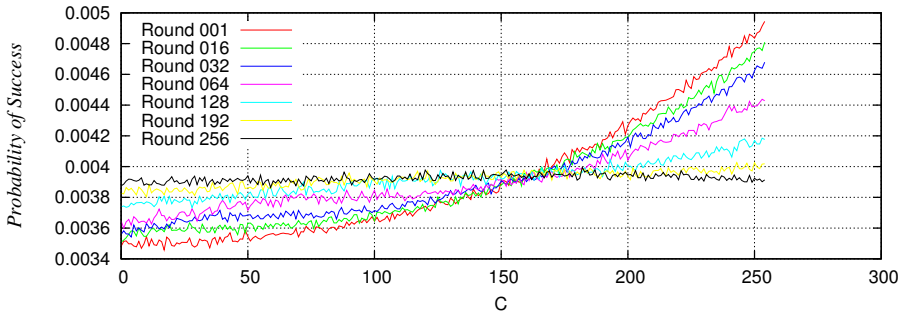


Figure 4.6: The probability of success of the biased correlation New_001, according the value C for some rounds of the PRGA.

elements of the vector $(j'_i, S'_i[j'_i], z_i)$ instead of $(j'_i, S'_i[i], S'_i[j'_i], z_i)$. This yields us all biases quite fast. This method can be easily generalized to the case when $S'_i[i]$ is also involved and it is dramatically faster than exhaustive search.

It would be clearer if we explain how we compute the fast Fourier Transform. we need to compute

$$\hat{f}(c_1, c_2, c_3) = \sum_{x_1, x_2, x_3} \theta^{-(c_1 x_1 + c_2 x_2 + c_3 x_3)} f(x_1, x_2, x_3)$$

This can be computed deploying 3 update stages:

$$f_1(c_1, x_2, x_3) = \sum_{x_1} \theta^{-c_1 x_1} f(x_1, x_2, x_3)$$

$$\Downarrow$$

$$f_2(c_1, c_2, x_3) = \sum_{x_2} \theta^{-c_2 x_2} f_1(c_1, x_2, x_3)$$

$$\Downarrow$$

$$\hat{f}(c_1, c_2, c_3) = \sum_{x_3} \theta^{-c_3 x_3} f_2(c_1, c_2, x_3)$$

We first compute f_1 table using f and store it. Then using f_1, we compute f_2. Finally, using f_2 we compute $\hat{f}$. Hence, the complexity of this method for the triplet case is $3 \cdot 2^{32}$, while for

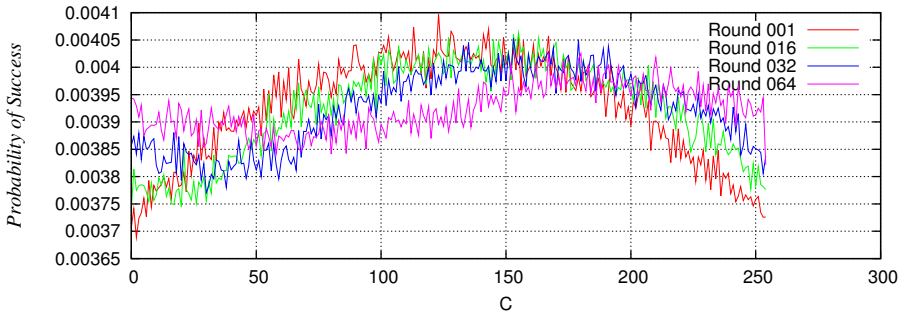


Figure 4.7: Probability of success of biased correlation New_002, according the value C for some rounds of the PRGA.

the exhaustive search the complexity reaches to 2^{47} if $\mathcal{N} = 10^7$, which is a reasonable number of samples we found out experimentally. We end up with all the biases for the triplet over $\mathbf{Z}_{256}^3$. Using this method, we found some new biases. We only list those which are significant and not artifact of known biases, those which can be bound with the biases of the KSA and can be exploited against WEP and WPA. They are listed in Figure 4.8. We do not list those biases which involve $K[0]$, $K[1]$ and $K[2]$, since they are already known in both WEP and WPA protocols. Interested readers can derive them all using this method. The complexity of the above method using the 4-tuple $(j_i', S_i'[i], S_i'[j_i'], z_i)$ is 2^{42} compared to 2^{55} of exhaustive search. Using this technique, we can recover all biases in $\mathbf{Z}_{256}^4$ as well in a reasonable amount time.

After investigation, it seems that all the listed biases in Figure 4.8 are artifact of a new conditional bias which is

$$\Pr[S_{16}'[j_{16}'] = 0 | z_{16} = -16] = P_{\mathsf{db}} = 0.038488$$

So far, we have no explanation about this new bias. We later modify this bias slightly and derive a new bias which we bind with the KSA weaknesses. An effort was done in [SGMPS11a] to prove similar biases, but the proof of this bias is still an open problem. In [SGMPS11a], the authors claimed that the number 16 in this bias is coming from the fact that we often set $\ell = 16$. They found a few more biases which depend on the size of the key. Finally, they launched a key length discovery attack on RC4 using those biases.

4.5 RC4 as a Black Box

The secret key words and the keystream words may be correlated if the weaknesses in the KSA and the PRGA can be bound. However, from an attacker's point of view, there is no reason to determine weaknesses in the KSA or the PRGA. Her objective is to find correlations between the known values and the secret key words. Moreover, the biased correlations previously found concern only elements inside a round of the PRGA. Correlations between elements from different rounds can not be highlighted.

In this section, we present another way to attack RC4. We consider RC4 as a black box. The objective is to discover linear correlations between the input (the secret key words) and the output (the keystream words), since we consider known keystream attacks. First, we study known RC4

j'_{16}	$S'_{16}[16]$	$S'_{16}[j'_{16}]$	z_{16}	C	Probability	Remark
0	0	1	1	240	$1.04/N$	New_010
0	0	1	50	224	$1.04/N$	New_011
0	0	1	68	192	$1.05/N$	New_012
0	0	1	98	224	$1.04/N$	New_013
0	0	1	148	192	$1.05/N$	New_014
0	0	1	162	224	$1.05/N$	New_015
0	0	1	186	96	$1.03/N$	New_016
0	0	1	187	80	$1.04/N$	New_017
0	0	1	251	80	$1.04/N$	New_018
0	0	2	19	208	$1.04/N$	New_019
0	0	2	127	16	$1.04/N$	New_020
0	0	2	147	208	$1.04/N$	New_021
0	0	2	255	16	$1.04/N$	New_022
0	0	4	59	80	$1.04/N$	New_023
0	0	4	123	80	$1.04/N$	New_024
0	0	8	19	208	$1.04/N$	New_025
0	0	8	55	144	$1.03/N$	New_026
0	0	8	81	240	$1.03/N$	New_027
0	0	8	215	144	$1.03/N$	New_028
0	0	8	241	48	$1.03/N$	New_029
0	0	8	243	208	$1.04/N$	New_030
0	0	32	39	144	$1.04/N$	New_031
0	0	32	191	16	$1.04/N$	New_032

Figure 4.8: correlations in the PRGA derived using the DFT algorithm.

correlations between the keystream words and the secret key words. Then, we propose a method to highlight new biases. Finally, we list the new discovered biases in RC4.

4.5.1 Maitra and Paul Correlation

In 2008, Maitra and Paul [MP08] discovered a new bias in RC4 which involves the state bytes and the keystream bytes from different rounds (see [MP08] for the proof).

$$z_{i+1} \overset{P_{\mathsf{Maitra}}(i)}{=} \frac{i(i+1)}{2} + \bar{K}[i]$$

where

$$P_{\mathsf{Maitra}}(i) = \begin{cases} \left(\frac{N-1}{N}\right)^2 \left(\frac{N-2}{N}\right)^{N-1} \cdot \gamma_1 + \frac{1}{N} & \text{if} \quad i = 0 \\ \left(\frac{N-1}{N}\right) \left(\frac{N-i-1}{N}\right) \left[\left(\frac{N-i}{N}\right) \left(\frac{N-1}{N}\right)^{\frac{i(i+1)}{2}+i+1} + \frac{1}{N}\right] \cdot & \text{if} \quad 1 < i < N \\ \left(\frac{N-2}{N}\right)^{N-i-1} \left(\frac{N-3}{N}\right)^{i-1} \cdot \gamma_{i+1} + \frac{1}{N} \end{cases}$$

and

$$\gamma_{i+1} = \frac{1}{N}\left(\frac{N-1}{N}\right)^{N-i-2} + \frac{1}{N}\left(\frac{N-1}{N}\right) - \frac{1}{N}\left(\frac{N-1}{N}\right)^{N-i-1}$$

Their bias does not hold for $i = 1$. This was experimentally observed by Maitra and Paul, but they did not provide any theoretical justification of it. We show in Sec. 5.1.3 why this is the case.

This bias has not been found with our previous technique, since these elements are not in the same round of the PRGA. However, with the black box technique, we are able to rediscover this bias and new ones.

We improve this bias in Chapter 5 by introducing new conditions to gain a better success probability together with extending it for the case when the values of the state bytes for some rounds of the KSA are known.

4.5.2 Discovering New Linear Correlations in RC4

We define a linear equation containing the input and output elements.

$$a_0 \cdot K[0] + \ldots + a_{\ell-1} \cdot K[\ell-1] + a_\ell \cdot z_1 + \ldots + a_{N+\ell-1} \cdot z_N \quad = \quad b \tag{4.4}$$

This kind of exhaustive search is identical to those presented in Sec. 4.4. First, we consider the subset of all a_i's defined by $A = \{-1, 0, 1\}$ and $b \in \mathbf{Z}/N\mathbf{Z}$. The number of equations is $N \cdot 3^{\ell+N} = 2^{439.11}$ for $N = 256$ and $\ell = 16$, which is obviously too large for an exhaustive search.

Based on the fact that $S_{N-1}[i]$ depends only on $K[0], \ldots, K[i-2], K[i-1]$ with a non negligible probability (Roos correlation), we can reduce the size of the equation by considering only the first ℓ keystream words. Hence, Eq. (4.4) becomes

$$a_0 \cdot K[0] + \ldots + a_{\ell-1} \cdot K[\ell-1] + a_\ell \cdot z_1 + \ldots + a_{2\ell-1} \cdot z_\ell = b$$

Thus, we obtain a number of equations equals to $N \cdot 3^{2\ell} = 2^{58.7}$ which is still too large for an exhaustive search. Thus, we reduced the secret keys length to $\ell = 5$ bytes to obtain $2^{23.8496}$ equations. We speculate that the correlations found with a RC4 key length of 5 bytes can be generalized to RC4 with a secret key of 16 bytes. Then, the supposed biased correlation are tested experimentally. After a few computation, we remarked that the constant value represented by b can be reduced to the subset generated by $i \cdot (i+1)/2$ with $i = 0, 1, 2 \ldots, 22$, since only the Roos correlation seems to be exploited in the KSA. Thus, the number of equations decreases to $23 \cdot 3^{10} = 2^{20.373}$. Figure 4.9 gives the correlations found in RC4 with a secret key of 5 bytes which were experimentally confirmed on RC4 with a key length of 16 bytes.

Equation	Probability	Remarks
$z_1 + K[0] + K[1] = 0$	1.35779/N	Klein
$z_1 - K[0] = 0$	1.11784/N	Maitra and Paul
$z_2 = 0$	2.01825/N	Mantin and Shamir
$z_2 + K[0] + K[1] + K[2] = -1$	1.36095/N	Klein
$z_1 - K[0] - K[1] = 1$	1.04237/N	New_bb_000
$z_1 - K[0] + K[1] = -1$	1.04969/N	New_bb_001
$z_3 + K[0] + K[1] + K[2] + K[3] = -3$	1.35362/N	Klein
$z_3 - K[0] + K[3] = -3$	1.04620/N	New_bb_002
$z_1 - K[0] - K[1] - K[2] = 3$	1.33474/N	Paul, Rathi and Maitra
$z_2 - K[0] - K[1] - K[2] = 3$	0.64300/N	New_bb_003
$z_3 - K[0] - K[1] - K[2] = 3$	1.13555/N	Maitra and Paul
$z_2 + K[1] + K[2] = -3$	1.36897/N	New_bb_004
$z_2 - K[1] - K[2] = 3$	1.36733/N	New_bb_005
$z_1 - K[2] = 3$	1.14193/N	New_bb_006
$z_1 + K[0] + K[1] - K[2] = 3$	1.14116/N	New_bb_007
$z_4 - K[0] + K[4] = 4$	1.04463/N	New_bb_008
$z_4 + K[0] + K[1] + K[2] + K[3] + K[4] = -6$	1.35275/N	Klein
$z_4 - K[0] - K[1] - K[2] - K[3] = 10$	1.11432/N	Maitra and Paul

Figure 4.9: The biased correlations experimentally observed with the black box technique with $\ell = 5$ in Eq. (4.5.2). Note that these biases are exploitable in RC4 with a secret key of 16 bytes as well.

Chapter 5

Unconditional Correlations in RC4, Exploitable against WEP and WPA

In this chapter, we are going to elaborate the unconditional biases in RC4. We only explain those which are exploitable against the WEP and the WPA protocols.

5.1 The Klein-Improved Attack

Andreas Klein combined the Jenkins correlation for the PRGA and weaknesses of the KSA and derived a correlation between the key bits and the keystream. This bias was further improved in [VV07] by recovering $\bar{K}[i]$'s instead of $K[i]$ to reduce the secret key bytes dependency. We use the theorem by Jenkins (see Theorem 4.1) and explain how it can be merged with the weaknesses of the KSA (see Algorithm 5.1). In fact, the attacker checks the conditions. If they all hold, she votes for $\bar{K}[i]$ using the key recovery relation.

Algorithm 5.1 The Klein-Improved Attack

Success Probability: P_{KI}
Assumptions: (see Figure 5.1)
1: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = S'_{i-1}[i] = S'_i[j'_i] = i - z_i$
Conditions: $(i - z_i) \notin \{S_t[t+1], \ldots, S_t[i-1]\}$ (Cond)
Key recovery relation: $\bar{K}[i] = S_t^{-1}[i - z_i] - \sigma_i(t)$

Exploiting the Jenkins correlation and the relations in the KSA and the PRGA, we obtain

1. $S'_i[j'_i] \overset{P_J}{=} i - z_i$ (Lemma 4.1)
2. $S'_i[j'_i] = S'_{i-1}[i]$
3. $S'_{i-1}[i] \overset{P_0}{=} S_i[i]$ (Lemma 3.4)
4. $S_i[i] = S_{i-1}[j_i]$

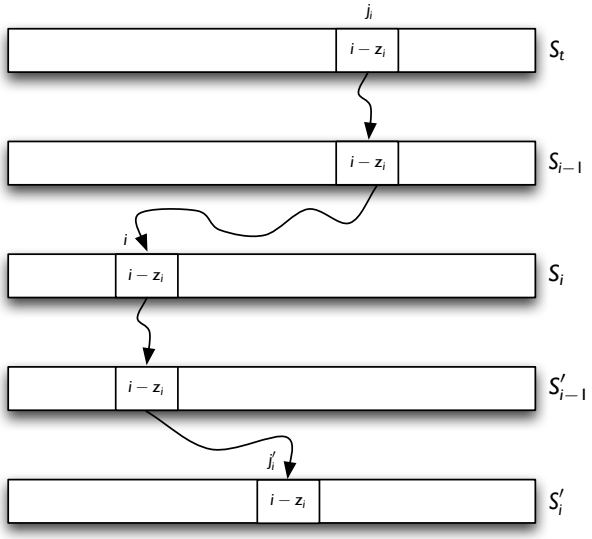


Figure 5.1: The RC4 state update in the Klein-Improved attack

5. $S_{i-1}[j_i] \overset{P_A^1}{\underset{\mathsf{Cond}'}{=}} S_t[j_i]$ (where Cond' is the event that $j_i \leq t$ or $j_i > i - 1$.)

6. $j_i = \bar{K}[i] + \sum_{x=1}^{i} S_{x-1}[x]$ (Lemma 3.3)

7. $\sum_{x=1}^{i} S_{x-1}[x] \overset{P_B}{=} \sigma_i$ (Lemma 3.4)

We make the same heuristic assumptions of independence as in Lemma 3.2, then using this Lemma and Lemma 3.4, we derive

$$P_{\mathrm{KI}}(i,t) = P_J \otimes P_0 \otimes P_A^1(i,t) \otimes P_B(i,t)$$

conditioned to Cond. Hence, the key recovery relation becomes

$$\bar{K}[i] \overset{P_{\mathrm{KI}}}{\underset{\mathsf{Cond}}{=}} S_t^{-1}[i - z_i] - \sigma_i(t)$$

5.1.1 SVV_008 Attack

The probability of this bias is smaller than the one used in the Klein-Improved attack and concerns only rounds $i \bmod 16 = 0$ of the PRGA. This bias turned out to be non-exploitable against WEP, since its probability is so low due to the fact that we only know up to state S_2. On the other hand, if up to state S_{i-1} is known, the bias is quite effective to recover $\bar{K}[i]$.

Similar to the Klein-Improved attack (see Algorithm 5.2), we obtain

Algorithm 5.2 The SVV_008 Attack

Success Probability: $P_{8\mathrm{I}}$
Assumptions: (see Figure 5.2)
1: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = S'_{i-1}[i] = S'_i[j'_i] = i + z_i$
Conditions: $(i + z_i) \notin \{S_t[t+1], \ldots, S_t[i-1]\}$ and $i = 16i'$ (Cond)
Key recovery relation: $\bar{K}[i] = S_t^{-1}[i + z_i] - \sigma_i(t)$

1. $S'_i[j'_i] \overset{P_8}{=} i + z_i$ (Figure 5 in [SVV10])
2. $S'_i[j'_i] = S'_{i-1}[i]$
3. $S'_{i-1}[i] \overset{P_0}{=} S_i[i]$ (Lemma 3.4)
4. $S_i[i] = S_{i-1}[j_i]$
5. $S_t[j_i] \underset{\mathsf{Cond}'}{\overset{P_A^1}{=}} S_{i-1}[j_i]$ (where Cond′ is the event that $j_i \leq t$ or $j_i > i - 1$)
6. $j_i = \bar{K}[i] + \sum_{x=1}^{i} S_{x-1}[x]$ (Lemma 3.3)
7. $\sum_{x=1}^{i} S_{x-1}[x] \overset{P_B}{=} \sigma_i$ (Lemma 3.4)

where $P_8 = \frac{1.05}{N}$ as in [SVV10]. Hence, the key recovery relation becomes

$$\bar{K}[i] \underset{\mathsf{Cond}'}{\overset{P_{8\mathrm{I}}}{=}} S_t^{-1}[i + z_i] - \sigma_i(t)$$

Using Lemma 3.2 and Lemma 3.4, its probability of success is $P_{8\mathrm{I}}(i, t)$, defined by

$$P_{8\mathrm{I}}(i, t) = P_8 \otimes P_0 \otimes P_A^1(i, t) \otimes P_B(i, t)$$

for any $(i + z_i) \notin \{S_t[t+1], \ldots, S_t[i-1]\}$, $i = 0 \bmod 16$.

5.1.2 The SVV_009 Attack

This bias is similar to the previous one and concerns only rounds $i \bmod 16 = 0$.

Algorithm 5.3 The SVV_009 Attack

Success Probability: $P_{9\mathrm{I}}$
Assumptions: (see Figure 5.3)
1: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = S'_{i-1}[i] = S'_i[j'_i] = -i - z_i$
Conditions: $(-i - z_i) \notin \{S_t[t+1], \ldots, S_t[i-1]\}$ and $i = 16i'$ (Cond)
Key recovery relation: $\bar{K}[i] = S_t^{-1}[-i - z_i] - \sigma_i(t)$

Using the same reasoning as SVV_008, this bias is not exploitable against WEP. On the other hand, if S_{i-1} is known, the bias is quite effective (see Algorithm 5.3).

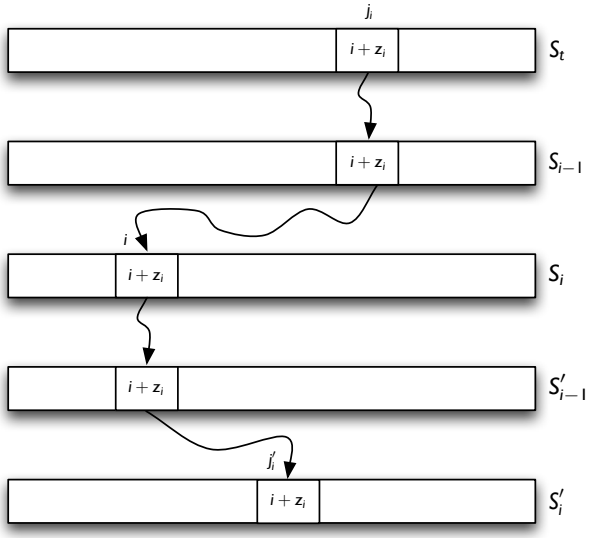


Figure 5.2: The RC4 state update in the SVV_008 attack

1. $S'_i[j'_i] \stackrel{P_9}{=} -i - z_i$ (Figure 5 in [SVV10])
2. $S'_i[j'_i] = S'_{i-1}[i]$
3. $S'_{i-1}[i] \stackrel{P_0}{=} S_i[i]$ (Lemma 3.4)
4. $S_i[i] = S_{i-1}[j_i]$
5. $S_t[j_i] \underset{\mathsf{Cond}'}{\stackrel{P_A^1}{=}} S_{i-1}[j_i]$ (where Cond' is the event that $j_i \leq t$ or $j_i > i - 1$.)
6. $j_i = \bar{K}[i] + \sum_{x=1}^{i} S_{x-1}[x]$ (Lemma 3.3)
7. $\sum_{x=1}^{i} S_{x-1}[x] \stackrel{P_B}{=} \sigma_i$ (Lemma 3.4)

where $P_9 = \frac{1.0338}{N}$ as in [SVV10]. Hence, the key recovery relation becomes

$$\bar{K}[i] \underset{\mathsf{Cond}}{\stackrel{P_{9\mathrm{I}}}{=}} S_t^{-1}[-i - z_i] - \sigma_i(t)$$

Using Lemma 3.2 and Lemma 3.4, its probability of success is $P_{9\mathrm{I}}(i,t)$, defined by

$$P_{9\mathrm{I}}(i,t) = P_9 \otimes P_0 \otimes P_A^1(i,t) \otimes P_B(i,t)$$

for any $(-i - z_i) \notin \{S_t[t+1], \ldots, S_t[i-1]\}$ and $i = 0 \bmod 16$.

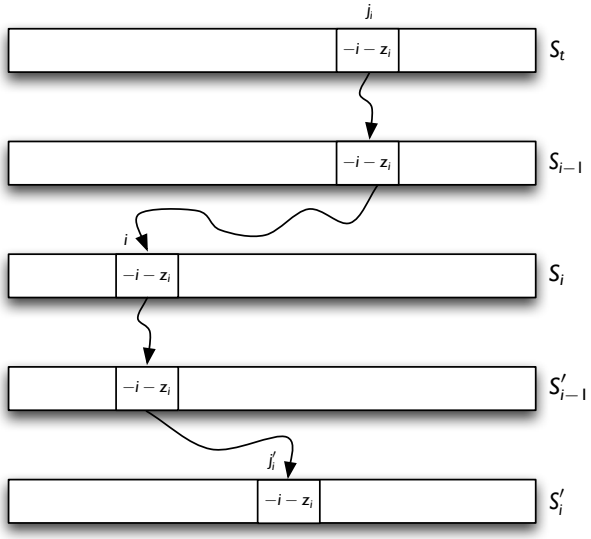


Figure 5.3: The RC4 state update in the SVV_009 attack

5.1.3 The Maitra-Paul-Improved Attack

Maitra and Paul illustrated in [MP08] that the $\mathsf{Pr}[z_{i+1} = j_i]$ is not uniformly distributed. We can use this bias to perform a key recovery attack on RC4 using Lemma 3.3. There was initially no condition on this bias, except that it does not hold for $i = 1$. Maitra and Paul observed this abnormality for $i = 1$ experimentally. We introduce some extra conditions which improve the success probability of this attack. In the following, we specify the assumptions on this attack and extract its probability of success and prove that the bias does not hold for $i = 1$. Their bias is directly exploitable for $t = 2$. We generalize it to any t. We do not use this bias for the WEP or the WPA attack, since it does not provide anything significantly more than what the Klein-Improved and the Korek attacks offer.

Algorithm 5.4 The Maitra-Paul-Improved Attack

Success Probability: P_{MPI}
Assumptions: (see Figure 5.4)
1: $(\exists m > i \mid j_m = i)$
2: $j_i = S_{-1}[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{m-1}[i] = S_m[m] = \cdots = S'_{i+1}[m]$
3: $m = S_{-1}[m] = \cdots = S_{m-1}[m] = S_m[i] = \cdots = S'_{i-1}[i] = S'_i[j'_i] = S'_{i+1}[i+1]$
4: $S'_i[i+1] = S'_{i+1}[j'_{i+1}] = 0$
4: **Conditions:** $i \neq 1$, $z_{i+1} \geq i$ and $(\forall 0 \leq i' \leq t : j_{i'} \neq z_{i+1})$
Key recovery relation: $\bar{K}[i] = z_{i+1} - \sigma_i$

Later, for the attack to work, an m should exist such that $j_m = i$. Due to the assumptions, we also assume $S_{-1}[m]$ and $S_{-1}[j_i]$ are maintained until S_{i-1}. Hence, we compute

$$\mathsf{Pr}[S_{-1}[j_i] = \cdots = S_{i-1}[j_i], S_{-1}[m] = \cdots = S_{i-1}[m]] = P_A^2(i, 0)$$

Furthermore, $j_i \geq i$ and $m \geq i$, otherwise they would both be swapped in the i-th initial KSA state updates. Right now, we fix m and then we sum over it. Thus, we compute

$$\mathsf{Pr}[j_i \geq i] = \left(\tfrac{N-i}{N}\right) \qquad \text{and} \qquad \mathsf{Pr}[j_m = i] = \left(\tfrac{1}{N}\right)$$

At the i-th stage of the KSA update, $S_{i-1}[j_i]$ is moved to $S_i[i]$. Due to the assumptions, $S_{i-1}[m]$ is maintained. So, we obtain $S_i[m] = m$. This holds with probability

$$\mathsf{Pr}[S_{i-1}[m] = S_i[m]] = P_A^1(i+1, i-1)$$

At the next stage, due to the assumptions, $S_i[i]$ and $S_i[m]$ are maintained until S_{m-1}, so at S_{m-1} we have $S_{m-1}[m] = m$ and $S_{m-1}[i] = j_i$. Thus, we compute

$$\mathsf{Pr}[S_i[i] = \cdots = S_{m-1}[i], S_i[m] = \cdots = S_{m-1}[m]] = P_A^2(m, i)$$

Since $j_m = i$, at the next update, we gain $S_m[m] = j_i$ and $S_m[i] = m$. Due to the assumptions, these two bytes are maintained until S'_{i-1}. So, we compute

$$\mathsf{Pr}[S_m[i] = \cdots = S'_{i-1}[i], S_m[m] = \cdots = S'_{i-1}[m]] = P_A^2(N+i-1, m)$$

At the i-th step of the PRGA update, $S'_{i-1}[i]$ is moved to $S'_i[j'_i]$ and due to the assumptions, we assume the value of $S'_{i-1}[m]$ is maintained. At this stage, we have $S'_i[m] = j_i$ and $S'_i[j'_i] = m$. We also have $S'_i[i+1] = 0$, due to the assumptions. Thus, we compute

$$\mathsf{Pr}[S'_{i-1}[m] = S'_i[m]] = P_A(N+i, N+i-2) \qquad \text{and} \quad \mathsf{Pr}[S'_i[i+1] = 0] = \left(\tfrac{1}{N}\right)$$

Finally, at the $(i+1)$-th PRGA update, $S'_i[i+1]$ is moved to $S'_{i+1}[j'_i]$ and $S'_i[j'_i]$ is moved to $S'_{i+1}[i+1]$. Due to the assumptions, the value of $S'_i[m]$ is maintained until S'_{i+1}. Ultimately, we compute

$$\mathsf{Pr}[S'_i[m] = S'_{i+1}[m]] = P_A^1(N+i+1, N+i-1)$$

We also know that

1. $z_{i+1} = S'_{i+1}[S'_i[i+1] + S'_i[j'_{i+1}]]$
2. $j'_{i+1} = j'_i + S'_i[i+1] = j'_i$
3. $S'_i[j'_{i+1}] = S'_i[j'_i] = S'_{i-1}[i] = S_m[i] = S_m[j_m] = S_{m-1}[m] = m$

Hence,

$$z_{i+1} = S'_{i+1}[m] = S_m[m] = S_{m-1}[j_m] = S_{m-1}[i] = S_i[i] = S_{i-1}[j_i] = j_i$$

So overall, using Lemma 3.1 and Corollary 3.1, we obtain

$$P_{\mathrm{MPI}}(i,t) = \mathsf{Pr}[\bar{K}[i] = z_{i+1} - \sigma_i] = P_B \otimes P_D = P_D(i)P_B(i,t) + \frac{1}{N-1}\left(1 - P_D(i)\right)\left(1 - P_B(i,t)\right)$$

Table 5.1: The unconditional biases for RC4, exploitable against WEP and WPA

row	reference	$\bar{f}$	$\bar{g}$	p
i	Klein − Improved	$S_t^{-1}[-z_i + i] - \sigma_i$	$(i - z_i) \notin \{S_t[t+1], \ldots, S_t[i-1]\}$	$P_{\mathrm{KI}}(i,t)$
$i \neq 1$	MP − Improved	$z_{i+1} - \sigma_i$	$i \neq 1$, $z_{i+1} \geq i$, $(\forall 0 \leq i' \leq t : j'_i \neq z_{i+1})$	$P_{\mathrm{MPI}}(i,t)$
$i = 16i'$	SVV_008	$z_i + i - \sigma_i$	$(i + z_i) \notin \{S_t[t+1], \ldots, S_t[i-1]\}$	$P_{8\mathrm{I}}(i,t)$
$i = 16i'$	SVV_009	$-z_i - i - \sigma_i$	$(-i - z_i) \notin \{S_t[t+1], \ldots, S_t[i-1]\}$	$P_{9\mathrm{I}}(i,t)$

where

$$P_D(i) = \mathsf{Pr}[z_{i+1} = j_i] = \sum_{m=i+1}^{N-1} \left(\frac{1}{N}\right)^2 \left(\frac{N-i}{N}\right) \cdot P_A^2(i,0) \cdot P_A^1(i+1, i-1) \cdot P_A^2(m,i) \cdot$$

$$P_A^2(N+i-1, m) \cdot P_A^1(N+i, N+i-2) \cdot P_A^1(N+i+1, N+i-1)$$

$$= \frac{(N-i-1)(N-i)}{N^3} \left(\frac{N-2}{N}\right)^{N-3+i} \left(\frac{N-1}{N}\right)^3$$

We added extra conditions to this attack. Clearly, $S_{i-1}[j_i] = j_i$ implies that $j_i \geq i$ and $\forall i' < i : j'_i \neq j_i$. So, we can have a better probability with conditions $z_{i+1} \geq i$ and $\forall i' \leq t : j_{i'} \neq z_{i+1}$.

This bias does not hold when $i = 1$. This is because, at the first iteration of the PRGA, we have $i = 1$ and $j'_1 = S_{N-1}[1] = m$. Then, we swap $S_{N-1}[1]$ and $S_{N-1}[m]$. Thus, we have $S'_1[1] = j_i$ and $S'_1[m] = m$. At the next iteration, $i = 2$ and $j'_2 = m + S'_1[2] = m$, so we swap $S'_1[2]$ and $S'_1[m]$. Finally, z_2 is computed as $z_2 = S'_2[S'_2[2] + S'_2[m]] = S'_2[m] = 0$. As a result, $z_2 \neq j_i$. The value $S[m]$ should be maintained during all steps of the KSA and the PRGA, while if $i = 1$, it would be swapped at the first stage.

5.2 Computation of the Biases

To summarize, all the biases in this chapter together with their corresponding probabilities are listed below and in Table 5.1.

$$\begin{aligned}
P_{\mathrm{KI}}(i,t) &= P_J \otimes P_0 \otimes P_A^1(i,t) \otimes P_B(i,t) \\
P_{\mathrm{MPI}}(i,t) &= P_D(i) \otimes P_B(i,t) \\
P_{8\mathrm{I}}(i,t) &= P_8 \otimes P_0 \otimes P_A^1(i,t) \otimes P_B(i,t) \\
P_{9\mathrm{I}}(i,t) &= P_9 \otimes P_0 \otimes P_A^1(i,t) \otimes P_B(i,t)
\end{aligned}$$

where $P_J = \frac{2}{N}$, $P_0 = \left(\frac{N-1}{N}\right)^{N-2}$, $P_8 = \frac{1.05}{N}$ and $P_9 = \frac{1.0338}{N}$.

$$\begin{aligned}
P_A^b(i,t) &= \left(\tfrac{N-b}{N}\right)^{i-t-1} \\
P_B(i,t) &= \textstyle\prod_{k=0}^{i-t-1} \left(\tfrac{N-k}{N}\right) + \tfrac{1}{N}\left(1 - \textstyle\prod_{k=0}^{i-t-1} \left(\tfrac{N-k}{N}\right)\right) \\
P_D(i) &= \tfrac{(N-i-1)(N-i)}{N^3} \left(\tfrac{N-2}{N}\right)^{N-3+i} \left(\tfrac{N-1}{N}\right)^3
\end{aligned}$$

These formulas are new. The biases were originally provided with probabilities for $t = -1$.

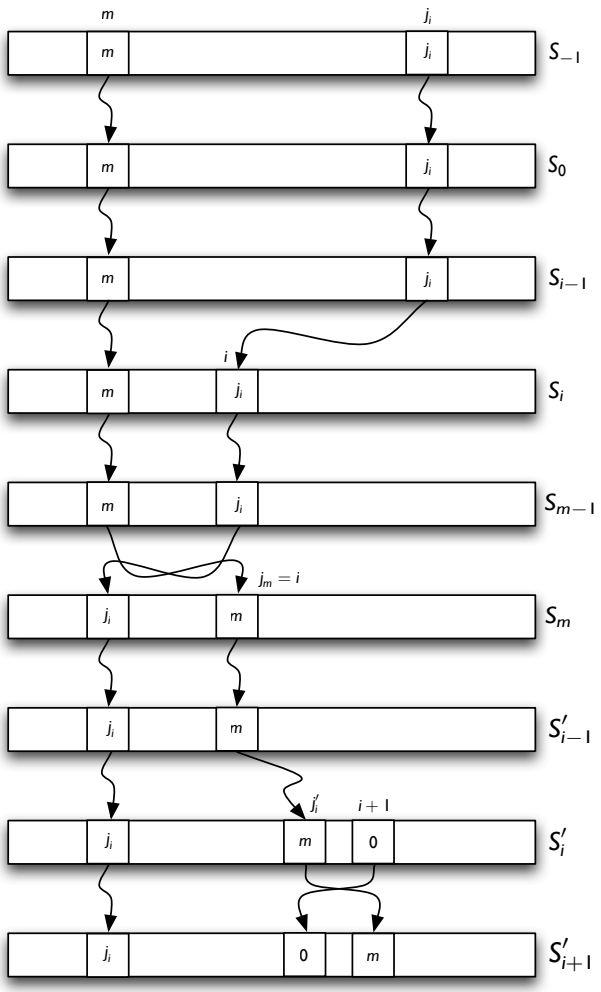


Figure 5.4: The RC4 state update in the Maitra-Paul-Improved attack

Chapter 6

Conditional Correlations, Exploitable against WEP and WPA

In this chapter, we are going to elaborate the conditional biases in RC4. We only explain those which are exploitable against WEP and WPA protocols.

6.1 The Sepehrdad-Vaudenay-Vuagnoux Bias

We showed in Chapter 4 that $\mathsf{Pr}[S'_{16}[j'_{16}] = 0 | z_{16} = -16]$ is not uniformly distributed and it holds with probability $P_{\text{db}} = 0.038488$. This bias was further analyzed but was not proved in [SGMPS11a]. Moreover, it was deployed to perform a key length discovery attack on RC4.

Algorithm 6.1 The Sepehrdad-Vaudenay-Vuagnoux Attack

Success Probability: $P_{\mathsf{SVV10}}(t)$
Assumptions: (see Figure 6.1)
1: $S_t[j_{16}] = \cdots = S_{15}[j_{16}] = S_{16}[16] = 0$
2: $i = 16$
Conditions: $S_t^{-1}[0] < t+1$ or $S_t^{-1}[0] > 15$, $z_{16} = -16$
Key recovery relation: $\bar{K}[16] = (S_t^{-1}[0] - \sigma_{16})$

Using the SVV_10 bias, the overall probability of the correlation between the keystream bytes and the key bytes is not easily computable. Therefore, we refined this bias and derived a new one $\mathsf{Pr}[S_{16}[16] = 0 | z_{16} = -16] = P_{\text{db2}} = 0.03689$. Next, merging this bias with the weaknesses of the KSA, we obtain

$$0 \underset{\mathsf{Cond}}{\overset{P_{\text{db2}}}{=}} S_{16}[16] = S_{15}[j_{16}] \overset{P_A^1(16,t)}{=} S_t[j_{16}] \qquad \text{and} \qquad j_{16} \overset{P_B}{=} \bar{K}[16] + \sigma_{16}$$

where $j_{16} \notin \{t+1, \ldots, 15\}$, due to Lemma 3.2. We should set $S_t^{-1}[0] < t+1$ or $S_t^{-1}[0] > 15$ to make sure that the index of zero is not trivially picked in the next iterations (see Algorithm 6.1). Using Lemma 3.2, we obtain

$$\bar{K}[16] \overset{P_{\mathsf{SVV10}}(t)}{=} S_t^{-1}[0] - \sigma_{16}$$

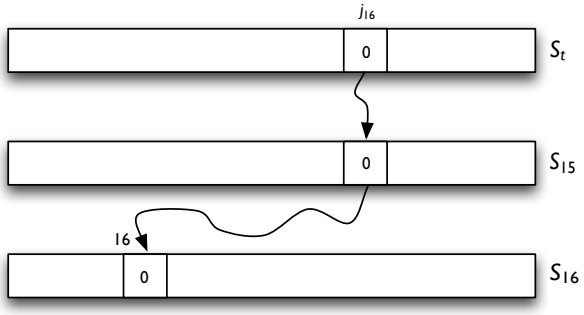


Figure 6.1: The RC4 state update in the SVV_10 attack

which holds with overall probability of

$$P_{\mathsf{SVV10}}(t) = P_{\mathsf{db2}} \otimes P_A^1(16, t) \otimes P_B(16, t)$$

6.2 The Korek Attacks

6.2.1 Introduction

In 2004, a hacker nicknamed Korek found 20 key recovery attacks on RC4 similar to the FMS attack [FMS01]. Korek classified them into three categories. The first class of attacks uses only z_1 and the state of the array S_{i-1} (i.e., $K[0], K[1] \ldots, K[i-1]$) of the KSA to recover the secret key $K[i]$. The second class of attacks uses the second byte of the keystream z_2. Ultimately, the last one highlights the improbable secret key bytes. They are called *negative attacks* or *impossible attacks*. We only mention 19 of such correlations, as the conditions of the attack A_u5_4 are rarely held in practice except for $i = 6$ and $t = 2$. Even for this case, its success probability is very close to the uniform distribution.

The Korek biases were used a few times in the literature for cryptanalysis of WEP [VV07, TWP07, BT09, DO11]. The only effort to analyze such biases were done in [Cha06, Vua10], but the success probability of such attacks were not computed precisely and the attack scenarios were not elaborated clearly. In this chapter, we are going to explain the theory behind all such biases and we will prove each of such correlations individually. During this process, we discovered that a few conditions set by Korek initially are not precise either, so we refined all those conditions to provide a higher success probability.

It will be discussed in the following chapters that the Korek biases are extremely important in cryptanalysis of WEP. We also demonstrate that this is not the case for WPA.

As all such biases are conditional, we introduce the conditions in which the attack holds. Then, we mention the assumptions we make and compute the probability that the assumptions hold. Finally, the key recovery relation is derived and the overall success probability of the attack is computed.

6.2.2 The A_u15 Attack

This attack is the best Korek attack with the highest success probability. First, we introduce the conditions for this attack to succeed, the assumptions we make and the equation for the key recovery.

Algorithm 6.2 The A_u15 attack

Success Probability: $P^1_{\mathsf{fixed}-j}$
Assumptions: (see Figure 6.2)
1: $S_t[i] = \cdots = S_{i-1}[i]$
2: $S_i[2] = \cdots = S_{N-1}[2] = S'_1[2] = 0$
3: $j_i = 2$
Conditions: $S_t[i] = 0$ and $z_2 = 0$
Key recovery relation: $\bar{K}[i] = 2 - \sigma_i$

We classify the conditions as

$$\mathsf{C_1} : S_t[i] = 0 \qquad \text{and} \quad \mathsf{C_2} : z_2 = 0$$

We also classify the assumptions and the events and the key recovery bias as

$$\begin{cases} \mathsf{S_1} : S_t[i] = \cdots = S_{i-1}[i] \\ \mathsf{S_2} : S_i[2] = \cdots = S_{N-1}[2] = S'_1[2] \\ \mathsf{S_3} : \bar{K}[i] = j_i - \sigma_i \\ \mathsf{E_1} : j_i = 2 \\ \mathsf{B} : \bar{K}[i] = 2 - \sigma_i \end{cases}$$

Then, we compute the theoretical success probability of the attack. The goal is to estimate the probability $\mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}]$. So, we compute

$$\begin{aligned} \mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}] &= \mathsf{Pr}[\mathsf{E_1}\mathsf{S_3}|\mathsf{C}] + \mathsf{Pr}[\mathsf{B}\neg\mathsf{S_3}|\mathsf{C}] \\ &= \mathsf{Pr}[\mathsf{E_1}|\mathsf{S_3}\mathsf{C}] \;.\; \mathsf{Pr}[\mathsf{S_3}|\mathsf{C}] + \mathsf{Pr}[\mathsf{B}|\neg\mathsf{S_3}\mathsf{C}] \;.\; (1 - \mathsf{Pr}[\mathsf{S_3}|\mathsf{C}]) \end{aligned}$$

Now,

$$\begin{aligned} \mathsf{Pr}[\mathsf{B}|\neg\mathsf{S_3}\mathsf{C}] &= \mathsf{Pr}[\mathsf{B}\neg\mathsf{E_1}|\neg\mathsf{S_3}\mathsf{C}] \\ &\approx \mathsf{Pr}[\mathsf{B}\neg\mathsf{E_1}|\mathsf{C}] \\ &= \mathsf{Pr}[\mathsf{B}|\neg\mathsf{E_1}\mathsf{C}] \;.\; \mathsf{Pr}[\neg\mathsf{E_1}|\mathsf{C}] \\ &\approx \tfrac{1}{N-1}\,(1 - \mathsf{Pr}[\mathsf{E_1}|\mathsf{C}]) \end{aligned}$$

Overall,

$$\begin{aligned} \mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}] &\approx \mathsf{Pr}[\mathsf{E_1}|\mathsf{C}] \;.\; \mathsf{Pr}[\mathsf{S_3}|\mathsf{C}] + \left(\tfrac{1-\mathsf{Pr}[\mathsf{E_1}|\mathsf{C}]}{N-1}\right) \;.\; (1 - \mathsf{Pr}[\mathsf{S_3}|\mathsf{C}]) \\ &= \mathsf{Pr}(\mathsf{E_1}|\mathsf{C}) \;.\; \left(\tfrac{N\mathsf{Pr}[\mathsf{S_3}|\mathsf{C}]-1}{N-1}\right) + \left(\tfrac{1-\mathsf{Pr}[\mathsf{S_3}|\mathsf{C}]}{N-1}\right) \end{aligned}$$

We then approximate $\mathsf{Pr}[\mathsf{S_3}|\mathsf{C}] \approx P_B(i,t)$ and we also have

$$\begin{aligned} \mathsf{Pr}[\mathsf{E_1}|\mathsf{C}] &= \mathsf{Pr}(\mathsf{C_1}|\mathsf{E_1}\mathsf{C_2}) \left(\tfrac{\mathsf{Pr}(\mathsf{E_1}|\mathsf{C_2})}{\mathsf{Pr}(\mathsf{C_1}|\mathsf{C_2})}\right) \\ &\approx \mathsf{Pr}(\mathsf{C_1}|\mathsf{E_1}\mathsf{C_2}) \\ &= \mathsf{Pr}(\mathsf{C_1}\mathsf{S_1}\mathsf{S_2}|\mathsf{E_1}\mathsf{C_2}) + \mathsf{Pr}(\mathsf{C_1}\neg(\mathsf{S_1}\mathsf{S_2})|\mathsf{E_1}\mathsf{C_2}) \\ &\approx \mathsf{Pr}(\mathsf{C_1}\mathsf{S_1}\mathsf{S_2}|\mathsf{E_1}\mathsf{C_2}) + \tfrac{1}{N}(1 - \mathsf{Pr}(\mathsf{S_1}\mathsf{S_2}|\mathsf{E_1}\mathsf{C_2})) \\ &\approx \mathsf{Pr}(\mathsf{C_1}\mathsf{S_1}\mathsf{S_2}|\mathsf{E_1}\mathsf{C_2}) + \tfrac{1}{N}\left(1 - P^1_A(i,t) \;.\; \left(\tfrac{N-1}{N}\right)^{N-i}\right) \end{aligned}$$

$$\begin{aligned}\mathsf{Pr}[\mathsf{C_1S_1S_2}|\mathsf{E_1C_2}] &= \left(\frac{\mathsf{Pr}[\mathsf{C_1S_1S_2E_1}|\mathsf{C_2}]}{\mathsf{Pr}[\mathsf{E_1}|\mathsf{C_2}]}\right) \\ &= \mathsf{Pr}[\mathsf{C_2}|\mathsf{C_1S_1S_2E_1}] \;.\; \left(\frac{\mathsf{Pr}[\mathsf{C_1S_1S_2E_1}]}{\mathsf{Pr}[\mathsf{C_2}] \;.\; \mathsf{Pr}[\mathsf{E_1}|\mathsf{C_2}]}\right)\end{aligned}$$

Deploying Lemma 4.3, we obtain

$$\mathsf{Pr}[\mathsf{C_1S_1S_2}|\mathsf{E_1C_2}] = \frac{1}{2}P_A^1(i,t)\left(\frac{N-1}{N}\right)^{N-i}$$

Therefore, overall we have

$$\begin{aligned}\mathsf{Pr}[\mathsf{B}|\mathsf{C_1C_2}] &= \left(\frac{NP_B(i,t)-1}{N-1}\right) \\ &\cdot \left[\frac{1}{2}P_A^1(i,t)\left(\frac{N-1}{N}\right)^{N-i} + \frac{1}{N}\left(1-P_A^1(i,t)\left(\frac{N-1}{N}\right)^{N-i}\right)\right] \\ &+ \left(\frac{1-P_B(i,t)}{N-1}\right)\end{aligned}$$

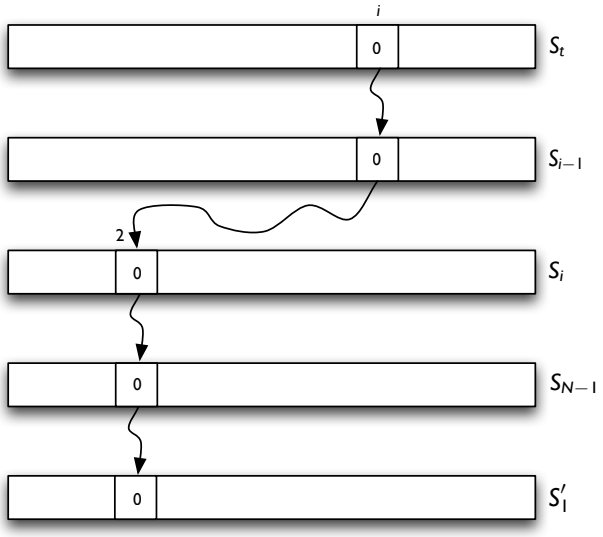


Figure 6.2: The RC4 state update in the A_u15 attack.

6.2.3 The A_s5_1 Attack

This attack in the generalization of the FMS attack. The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.3.

The attack works as follows: Assume $S_t[1] = \alpha$, $S_t[\alpha] = \beta$ and also assume $\alpha + \beta = i$ by the conditions. Then, assume these two values are maintained at the same position up to the state

Algorithm 6.3 The A_s5_1 attack

Success Probability: Kor_2^3
Assumptions: (see Figure 6.3)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1] = \alpha$
2: $S_t[\alpha] = \cdots = S_{i-1}[\alpha] = S_i[\alpha] = \cdots = S_{N-1}[\alpha] = \beta$
3: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i]$
Conditions: $S_t[1] < t+1$, $S_t[1]+S_t[S_t[1]] = i$, $z_1 \neq \{S_t[1], S_t[S_t[1]]\}$, ($S_t^{-1}[z_1] < t+1$ or $S_t^{-1}[z_1] > i-1$)
Key recovery relation: $\bar{K}[i] = S_t^{-1}[z_1] - \sigma_i$

S_{i-1} and then up to the state S_{N-1}. Another assumption we make is that $S_i[i]$ is maintained up to the state S_{N-1}. At the first iteration of the PRGA, $i = 1$ and $j'_1 = S_{N-1}[1] = \alpha$. Then, a swap is made between $S_{N-1}[1]$ and $S_{N-1}[\alpha]$. Finally, we have $z_1 = S'_1[S'_1[1] + S'_1[\alpha]] = S'_1[\alpha + \beta] = S'_1[i] = S_i[i] = S_{i-1}[j_i] = S_t[j_i]$. Hence, we obtain $z_1 = S_t[j_i]$ and so $j_i = S_t^{-1}[z_1]$. As we also have $\bar{K}[i] = j_i - \sigma_i(t)$, we conclude from the previous equation that $\bar{K}[i] = S_t^{-1}[z_1] - \sigma_i$. The last condition on z_1 in the list of conditions are for filtering out some incorrect events leading to the same results. This makes the success probability and the key recovery more precise. The condition $\{S_t[1], S_t^{-1}[z_1]\} < t+1$ is to make $\{\alpha, j_i\} < t+1$, therefore it is not trivially swapped during the KSA iterations. We also should make sure that $z_1 \neq \{\alpha, \beta\}$, so we end up with 3 elements in the state that have not moved. Thus, we need the condition $z_1 \neq \{S_t[1], S_t[S_t[1]]\}$.

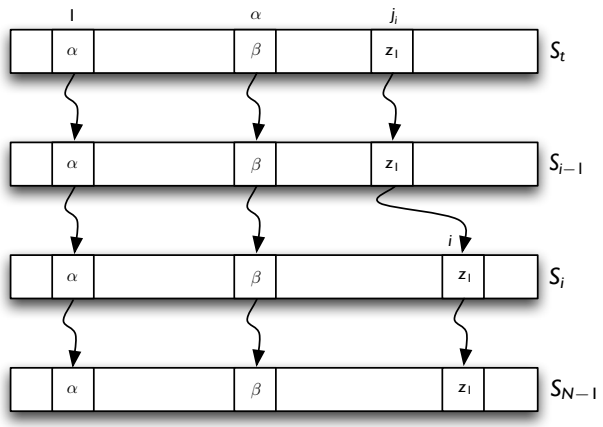


Figure 6.3: The RC4 state update in the A_s5_1 attack.

6.2.4 The A_s13 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.4.

In this attack, a nice event happens in the PRGA which automatically makes $S'_1[1] = 0$. Assume that $S_{N-1}[i] = \gamma$, then we explain that $\gamma = 0$. At the first step of the PRGA, $i = 1$ and $j'_1 =$

Algorithm 6.4 The A_s13 attack

Success Probability: Kor_1^2
Assumptions: (see Figure 6.4)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1] = i$
2: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i] = 0$
Conditions: $S_t[1] = i$, $(S_t^{-1}[0] < t + 1$ or $S_t^{-1}[0] > i - 1)$ and $z_1 = i$
Key recovery relation: $\bar{K}[i] = S_t^{-1}[z_1] - \sigma_i$

$S_{N-1}[1] = i$, so we swap $S_{N-1}[1]$ and $S_{N-1}[i]$. To compute z_1, we have $z_1 = S'_1[S'_1[1] + S'_1[i]] = S'_1[\gamma + i] = i$, since from the conditions we have $z_1 = i$. This makes us conclude that $\gamma = 0$. We already know from the relations in the KSA that $S_{i-1}[j_i] = S_i[i]$ and we assume that $S_{i-1}[j_i] = S_t[j_i]$ and also $S_{i-1}[j_i] = 0$. Thus, $S_t[j_i] = 0$. Then, we obtain $j_i = S_t^{-1}[0]$. Using the similar formulas as the previous attacks, we get $\bar{K}[i] = S_t^{-1}[0] - \sigma_i$.

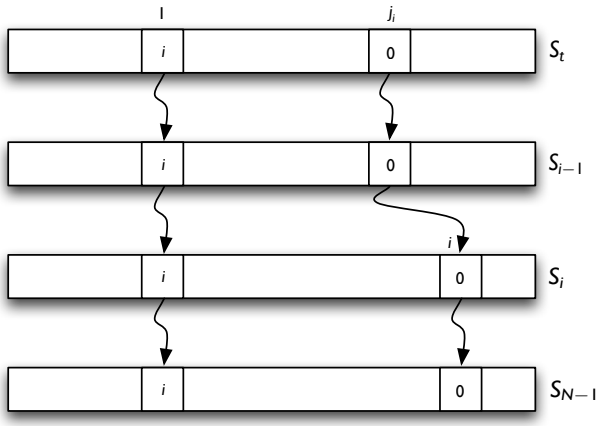


Figure 6.4: The RC4 state update in the A_s13 attack.

6.2.5 The A_u13_1 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.5.

Algorithm 6.5 The A_u13_1 attack

Success Probability: Kor_1^2
Assumptions: (see Figure 6.5)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1] = i$
2: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i] = 1 - i$
Conditions: $S_t[1] = i$, $(S_t^{-1}[1 - i] < t + 1$ or $S_t^{-1}[1 - i] > i - 1)$ and $z_1 = 1 - i$
Key recovery relation: $\bar{K}[i] = S_t^{-1}[z_1] - \sigma_i$

At the first step of the PRGA, $i = 1$ and $j'_1 = S_{N-1}[1] = i$, so we swap $S_{N-1}[1]$ and $S_{N-1}[i]$. To

compute z_1, we have $z_1 = S'_1[S'_1[1] + S'_1[i]] = S'_1[1] = 1 - i$. We already know from the relations in the KSA that $S_{i-1}[j_i] = S_i[i]$ and we assume that $S_{i-1}[j_i] = S_t[j_i]$ and also $S_{i-1}[j_i] = 1 - i$. Thus, $S_t[j_i] = 1 - i$. Then, we obtain $j_i = S_t^{-1}[1 - i]$. Using the similar formulas as the previous attacks, we get $\bar{K}[i] = S_t^{-1}[z_1] - \sigma_i$.

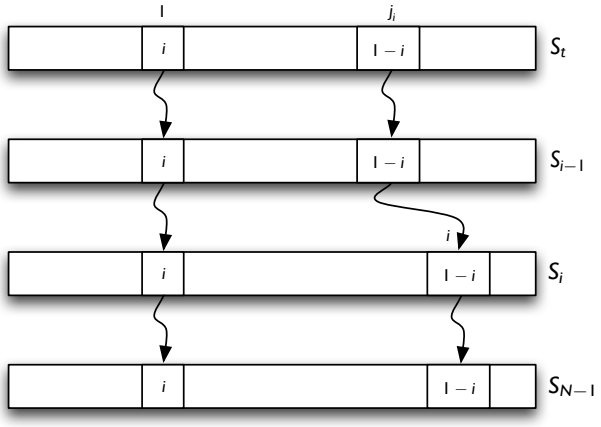


Figure 6.5: The RC4 state update in the A_u13_1 attack.

6.2.6 The A_u5_1 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.6.

Algorithm 6.6 The A_u5_1 attack

Success Probability: Kor_2^3
Assumptions: (see Figure 6.6)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1] = i$
2: assuming $S_t^{-1}[z_1] = \alpha$, we should have $S_t[\alpha] = \cdots = S_{i-1}[\alpha] = S_i[\alpha] = \cdots = S_{N-1}[\alpha] = z_1$.
3: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i] = S_t^{-1}[z_1] - i$
Conditions: $S_t[1] = i$, $S_t^{-1}[z_1] < t + 1$, $S_t^{-1}[S_t^{-1}[z_1] - i] \neq 1$, $(S_t^{-1}[S_t^{-1}[z_1] - i] < t + 1$ or $S_t^{-1}[S_t^{-1}[z_1] - i] > i - 1)$, $z_1 \neq \{i, 1 - i, S_t^{-1}[z_1] - i\}$ and $S_t^{-1}[z_1] \neq 2i$
Key recovery relation: $\bar{K}[i] = S_t^{-1}[S_t^{-1}[z_1] - i] - \sigma_i$

At the first stage of the PRGA, we have $i = 1$ and $j'_1 = S_{N-1}[1] = i$. So we swap $S_{N-1}[1]$ and $S_{N-1}[i]$. We know that $S_i[i] = S_{i-1}[j_i] = S_t^{-1}[z_1] - i$ and also $j_i = S_t^{-1}[S_t^{-1}[z_1] - i]$. The output z_1 would be $z_1 = S'_1[S'_1[1] + S'_1[i]] = S'_1[S_t^{-1}[z_1] - i + i] = S'_1[S_1^{-1}[z_1]] = z_1$. Therefore, we have $\bar{K}[i] = S_t^{-1}[S_t^{-1}[z_1] - i] - \sigma_i$. The condition $z_1 \neq \{i, 1 - i\}$ is to filter out the attacks A_u13_1 and A_s13. $S_t^{-1}[z_1] < t + 1$, because otherwise z_1 would be swapped in the next iterations of the KSA. If $S_t^{-1}[S_t^{-1}[z_1] - i] = 1$, then $j_i = 1$ and so a swap will be made between $S_{i-1}[1]$ and $S_{i-1}[i]$, in the i-th step of the KSA.

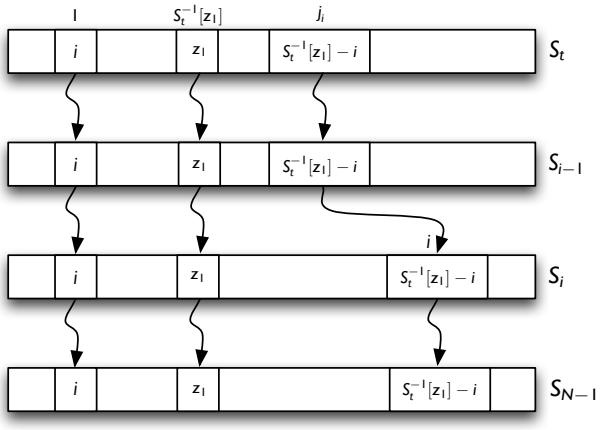


Figure 6.6: The RC4 state update in the A_u5_1 attack.

6.2.7 The A_u5_2 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.7.

Algorithm 6.7 The A_u5_2 attack

Success Probability: $P^2_{\mathsf{fixed}-j}$
Assumptions: (see Figure 6.7)
1: $S_t[i] = \cdots = S_{i-1}[i] = S_i[1] = \cdots = S_{N-1}[1] = 1$
2: $S_t[2] = \cdots = S_{N-1}[2] = z_1$
3: $j_i = 1$
Conditions: $S_t[i] = 1$ and $z_1 = S_t[2]$
Key recovery relation: $\bar{K}[i] = 1 - \sigma_i$

This is one of the attacks in which we assume $j_i = 1$. Lets check how the PRGA works. Initially, $i = 1$ and $j'_1 = S_{N-1}[1] = 1$. In the swap step, no swap is made, since we have to swap $S_{N-1}[1]$ and $S_{N-1}[1]$. Hence, $z_1 = S'_1[S'_1[1] + S'_1[1]] = S'_1[2] = S_t[2]$. So, the key recovery formula becomes $\bar{K}[i] = 1 - \sigma_i$.

We classify the conditions as

$$\mathsf{C}_1 : S_t[i] = 1 \qquad \text{and} \quad \mathsf{C}_2 : z_1 = S_t[2]$$

We also classify the assumptions and the events and the key recovery bias as

$$\begin{cases} \mathsf{S}_1 : S_t[i] = \cdots = S_{i-1}[i] \\ \mathsf{S}_2 : S_i[1] = \cdots = S_{N-1}[1] \\ \mathsf{S}_3 : S_t[2] = \cdots = S_{N-1}[2] \\ \mathsf{S}_4 : \bar{K}[i] = j_i - \sigma_i \\ \mathsf{E}_1 : j_i = 1 \\ \mathsf{B} : \bar{K}[i] = 1 - \sigma_i \end{cases}$$

Next, we compute the theoretical success probability of the attack. The goal is to estimate $\mathsf{Pr}[\mathsf{B}|\mathsf{C}_1, \mathsf{C}_2]$. Deploying a similar approach to the one of the attack A_u15, we end up with

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C}_1, \mathsf{C}_2] = \mathsf{Pr}(\mathsf{E}_1|\mathsf{C}) \; . \; \left(\frac{NP_B(i,t) - 1}{N-1}\right) + \left(\frac{1 - P_B(i,t)}{N-1}\right)$$

where

$$\mathsf{Pr}[\mathsf{E}_1|\mathsf{C}] \approx \mathsf{Pr}(\mathsf{C}_1\mathsf{S}_1\mathsf{S}_2\mathsf{S}_3|\mathsf{E}_1\mathsf{C}_2) + \frac{1}{N}\left(1 - P_A^2(i,t) \; . \; \left(\frac{N-2}{N}\right)^{N-i-1}\right)$$

$$\begin{aligned}\mathsf{Pr}[\mathsf{C}_1\mathsf{S}_1\mathsf{S}_2\mathsf{S}_3|\mathsf{E}_1\mathsf{C}_2] &= \left(\frac{\mathsf{Pr}[\mathsf{C}_1\mathsf{S}_1\mathsf{S}_2\mathsf{S}_3\mathsf{E}_1|\mathsf{C}_2]}{\mathsf{Pr}[\mathsf{E}_1|\mathsf{C}_2]}\right) \\ &= \mathsf{Pr}[\mathsf{C}_2|\mathsf{C}_1\mathsf{S}_1\mathsf{S}_2\mathsf{S}_3\mathsf{E}_1] \; . \; \left(\frac{\mathsf{Pr}[\mathsf{C}_1\mathsf{S}_1\mathsf{S}_2\mathsf{S}_3\mathsf{E}_1]}{\mathsf{Pr}[\mathsf{C}_2] \; . \; \mathsf{Pr}[\mathsf{E}_1|\mathsf{C}_2]}\right)\end{aligned}$$

As $\mathsf{Pr}[\mathsf{C}_2]$ is not uniformly distributed, we use Lemma 4.6 to compute its value. Then, we approximate $\mathsf{Pr}[\mathsf{C}_2] \approx \left(\frac{N-1}{N}\right)^{t-2} \; . \; \mathsf{Pr}[z_1 = \bar{K}[2] + 3]$. Deploying Lemma 4.6, we obtain

$$\begin{aligned}\mathsf{Pr}[\mathsf{C}_1\mathsf{S}_1\mathsf{S}_2\mathsf{S}_3|\mathsf{E}_1\mathsf{C}_2] &= \left(\frac{N}{N-1}\right)^{t-2} \; . \; \frac{N}{\xi}\left(\frac{1}{N} \; . \; \frac{1}{N}\left(\frac{N-2}{N}\right)^{N-1-i} \; . \; P_A^2(i,t)\right) \\ &= \frac{1}{N\xi}P_A^2(i,t)\left(\frac{N}{N-1}\right)^{t-2}\left(\frac{N-2}{N}\right)^{N-1-i}\end{aligned}$$

Therefore, overall we have

$$\begin{aligned}\mathsf{Pr}[\mathsf{B}|\mathsf{C}_1\mathsf{C}_2] &= \frac{1}{N}\left(\frac{NP_B(i,t) - 1}{N-1}\right) \\ &\quad . \quad \left[\frac{1}{\xi}P_A^2(i,t)\left(\frac{N}{N-1}\right)^{t-2}\left(\frac{N-2}{N}\right)^{N-1-i} + \left(1 - P_A^2(i,t)\left(\frac{N-2}{N}\right)^{N-i-1}\right)\right] \\ &\quad + \quad \left(\frac{1 - P_B(i,t)}{N-1}\right)\end{aligned}$$

6.2.8 The A_u13_2 Attack

This attack is very similar to the previous attack. The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.8.

Algorithm 6.8 The A_u13_2 attack

Success Probability: $P^3_{\mathsf{fixed}-j}$
Assumptions: (see Figure 6.8)
0: $S_t[1] = \cdots = S_{i-1}[1] = S_i[i] = \cdots = S_{N-1}[i] = 0$
0: $S_t[i] = \cdots = S_{i-1}[i] = S_i[1] = \cdots = S_{N-1}[1] = i$
Conditions: $S_t[i] = i$, $S_t[1] = 0$ and $z_1 = i$
Key recovery relation: $\bar{K}[i] = 1 - \sigma_i$

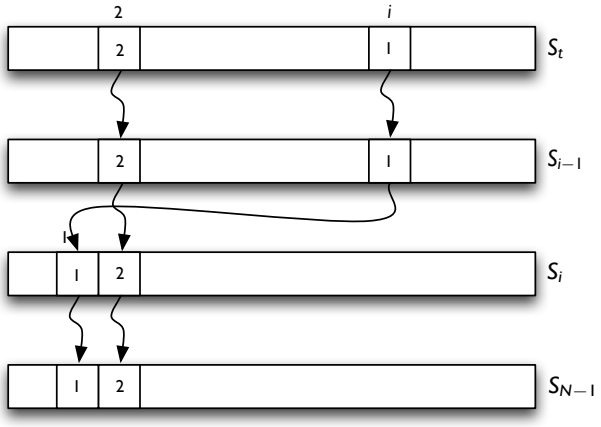


Figure 6.7: The RC4 state update in the A_u5_2 attack.

Again, we assume $j_i = 1$. In the KSA, we know that $S_{i-1}[1] = 0$ and $S_{i-1}[i] = i$. At the i-th stage, due to the swap, we have $S_i[1] = i$ and $S_i[i] = 0$. We assume these two values are maintained until the end of the KSA. In the PRGA, initially $i = 1$ and $j'_1 = S_{N-1}[1] = i$. So, we swap $S_{N-1}[1]$ and $S_{N-1}[i]$. Then, $z_1 = S'_1[S'_1[1] + S'_1[i]] = i$. Hence, the key recovery formula would be $\bar{K}[i] = 1 - \sigma_i$.

We classify the conditions as

$$\mathsf{C_1} : S_t[i] = i \qquad \text{and} \quad \mathsf{C_2} : S_t[1] = 0 \qquad \text{and} \qquad \mathsf{C_3} : z_1 = i$$

We also classify the assumptions and the events and the key recovery bias as

$$\begin{cases} \mathsf{S_1} : S_t[i] = \cdots = S_{i-1}[i] \\ \mathsf{S_2} : S_i[1] = \cdots = S_{N-1}[1] \\ \mathsf{S_3} : S_t[1] = \cdots = S_{i-1}[1] \\ \mathsf{S_4} : S_i[i] = \cdots = S_{N-1}[i] \\ \mathsf{S_5} : \bar{K}[i] = j_i - \sigma_i \\ \mathsf{E_1} : j_i = 1 \\ \mathsf{B} : \bar{K}[i] = 1 - \sigma_i \end{cases}$$

Now, we compute the theoretical success probability of the attack. The goal is to estimate $\mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}]$. Deploying a similar approach as the one of the attack A_u15, we end up with

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}] = \mathsf{Pr}(\mathsf{E_1}|\mathsf{C}) \ . \ \left(\frac{NP_B(i,t) - 1}{N-1}\right) + \left(\frac{1 - P_B(i,t)}{N-1}\right)$$

where

$$\mathsf{Pr}[\mathsf{E_1}|\mathsf{C}] \approx \mathsf{Pr}(\mathsf{C_1C_2S_1S_2S_3S_4}|\mathsf{E_1C_3}) + \frac{1}{N}\left(1 - P_A^2(i,t) \ . \ \left(\frac{N-2}{N}\right)^{N-i-1}\right)$$

$$\begin{aligned}
\mathsf{Pr}[\mathsf{C_1C_2S_1S_2S_3S_4}|\mathsf{E_1C_3}] &= \left(\frac{\mathsf{Pr}[\mathsf{C_1C_2S_1S_2S_3S_4E_1}|\mathsf{C_3}]}{\mathsf{Pr}[\mathsf{E_1}|\mathsf{C_3}]}\right) \\
&= \mathsf{Pr}[\mathsf{C_3}|\mathsf{C_1C_2S_1S_2S_3S_4E_1}] \;.\; \left(\frac{\mathsf{Pr}[\mathsf{C_1C_2S_1S_2S_3S_4E_1}]}{\mathsf{Pr}[\mathsf{C_3}] \;.\; \mathsf{Pr}[\mathsf{E_1}|\mathsf{C_3}]}\right) \\
&= \left(\frac{N-1}{N}\right)^{t+1} \left(\frac{N-2}{N}\right)^{N-1-i} \;.\; P_A^2(i,t)
\end{aligned}$$

$\mathsf{Pr}[\mathsf{C_3}]$ is uniformly distributed in this case and we also have

$$\mathsf{Pr}[S_t[i] = i] = \left(\frac{N-1}{N}\right)^{t+1}$$

Therefore, overall we have

$$\begin{aligned}
\mathsf{Pr}[\mathsf{B}|\mathsf{C_1C_2C_3}] &= \left(\frac{NP_B(i,t)-1}{N-1}\right) \\
&\;.\; \left[\left(\frac{N-1}{N}\right)^{t+1} \left(\frac{N-2}{N}\right)^{N-1-i} \;.\; P_A^2(i,t) + \frac{1}{N}\left(1 - P_A^2(i,t)\left(\frac{N-2}{N}\right)^{N-i-1}\right)\right] \\
&+ \left(\frac{1-P_B(i,t)}{N-1}\right)
\end{aligned}$$

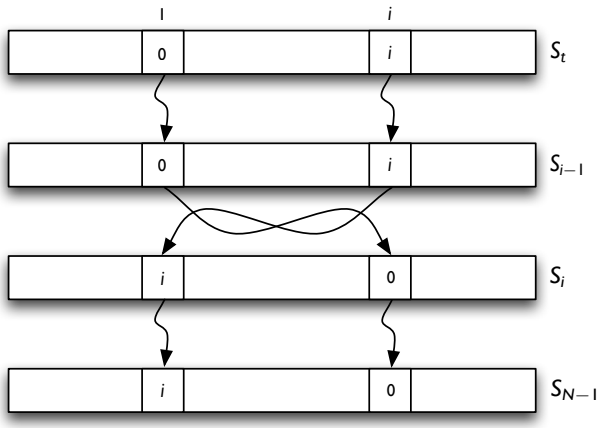


Figure 6.8: The RC4 state update in the A_u13_2 attack.

6.2.9 The A_u13_3 Attack

This attack is going through exactly the same approach as the previous attack, but with different values. The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.8.

Again, we assume $j_i = 1$. In the KSA, we know that $S_{i-1}[1] = 1 - i$ and $S_{i-1}[i] = i$. At the i-th stage, due to swap, we have $S_i[1] = i$ and $S_i[i] = 1 - i$. We assume these two values are maintained

Algorithm 6.9 The A_u13_3 attack

Success Probability: $P^3_{\mathsf{fixed}-j}$
Assumptions: (see Figure 6.9)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[i] = \cdots = S_{N-1}[i] = 1 - i$
2: $S_t[i] = \cdots = S_{i-1}[i] = S_i[1] = \cdots = S_{N-1}[1] = i$
3: $j_i = 1$
Conditions: $S_t[i] = i$, $S_t[1] = 1 - i$ and $z_1 = 1 - i$
Key recovery relation: $\bar{K}[i] = 1 - \sigma_i$

until the end of the KSA. In the PRGA, $i = 1$ and $j'_1 = S_{N-1}[1] = i$. So, we swap $S_{N-1}[1]$ and $S_{N-1}[i]$. Then, $z_1 = S'_1[S'_1[1] + S'_1[i]] = S'_1[1] = 1 - i$. Hence, the key recovery formula would be $\bar{K}[i] = 1 - \sigma_i$.

We classify the conditions as

$$\mathsf{C_1} : S_t[i] = i \qquad \text{and} \quad \mathsf{C_2} : S_t[1] = 1 - i \qquad \text{and} \qquad \mathsf{C_3} : z_1 = 1 - i$$

We also classify the assumptions and the events and the key recovery bias as

$$\begin{cases} \mathsf{S_1} : S_t[i] = \cdots = S_{i-1}[i] \\ \mathsf{S_2} : S_i[1] = \cdots = S_{N-1}[1] \\ \mathsf{S_3} : S_t[1] = \cdots = S_{i-1}[1] \\ \mathsf{S_4} : S_i[i] = \cdots = S_{N-1}[i] \\ \mathsf{S_5} : \bar{K}[i] = j_i - \sigma_i \\ \mathsf{E_1} : j_i = 1 \\ \mathsf{B} : \bar{K}[i] = 1 - \sigma_i \end{cases}$$

Now, we compute the theoretical success probability of the attack. The goal is to estimate $\mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}]$. Deploying a similar approach to the one of the attack A_u15, we end up with

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}] = \mathsf{Pr}(\mathsf{E_1}|\mathsf{C}) \; . \; \left(\frac{NP_B(i,t) - 1}{N - 1}\right) + \left(\frac{1 - P_B(i,t)}{N - 1}\right)$$

where

$$\mathsf{Pr}[\mathsf{E_1}|\mathsf{C}] \approx \mathsf{Pr}(\mathsf{C_1C_2S_1S_2S_3S_4}|\mathsf{E_1C_3}) + \frac{1}{N}\left(1 - P^2_A(i,t) \; . \; \left(\frac{N-2}{N}\right)^{N-i-1}\right)$$

$$\begin{aligned} \mathsf{Pr}[\mathsf{C_1C_2S_1S_2S_3S_4}|\mathsf{E_1C_3}] &= \left(\frac{\mathsf{Pr}[\mathsf{C_1C_2S_1S_2S_3S_4E_1}|\mathsf{C_3}]}{\mathsf{Pr}[\mathsf{E_1}|\mathsf{C_3}]}\right) \\ &= \mathsf{Pr}[\mathsf{C_3}|\mathsf{C_1C_2S_1S_2S_3S_4E_1}] \; . \; \left(\frac{\mathsf{Pr}[\mathsf{C_1C_2S_1S_2S_3S_4E_1}]}{\mathsf{Pr}[\mathsf{C_3}] \; . \; \mathsf{Pr}[\mathsf{E_1}|\mathsf{C_3}]}\right) \\ &= \left(\frac{N-1}{N}\right)^{t+1}\left(\frac{N-2}{N}\right)^{N-1-i} \; . \; P^2_A(i,t) \end{aligned}$$

$\mathsf{Pr}[\mathsf{C_3}]$ is uniformly distributed in this case and we also have

$$\mathsf{Pr}[S_t[i] = i] = \left(\frac{N-1}{N}\right)^{t+1}$$

Therefore, overall we have

$$\begin{aligned}\mathsf{Pr}[\mathsf{B}|\mathsf{C_1C_2C_3}] &= \left(\frac{NP_B(i,t)-1}{N-1}\right) \\ &\cdot \left[\left(\frac{N-1}{N}\right)^{t+1}\left(\frac{N-2}{N}\right)^{N-1-i} \cdot P_A^2(i,t) + \frac{1}{N}\left(1-P_A^2(i,t)\left(\frac{N-2}{N}\right)^{N-i-1}\right)\right] \\ &+ \left(\frac{1-P_B(i,t)}{N-1}\right)\end{aligned}$$

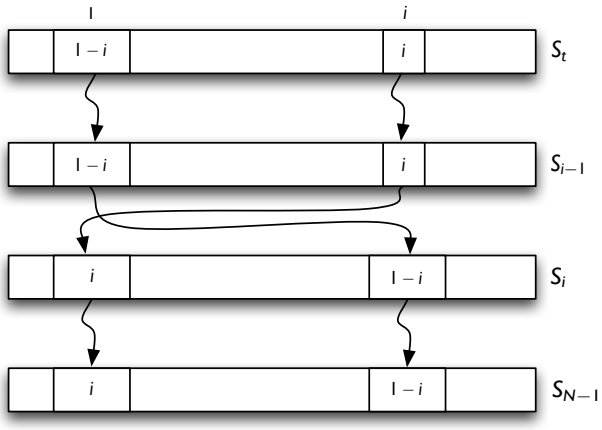


Figure 6.9: The RC4 state update in the A_u13_3 attack.

6.2.10 The A_u5_3 Attack

This attack is the extension of A_u13_2 and A_u13_3 attacks. The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.10.

Algorithm 6.10 The A_u5_3 attack

Success Probability: $P^5_{\mathsf{fixed}-j}$
Assumptions: (see Figure 6.10)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[i] = \cdots = S_{N-1}[i] = S_t^{-1}[z_1] - i$
2: $S_t[i] = \cdots = S_{i-1}[i] = S_i[1] = \cdots = S_{N-1}[1] = i$
3: $S_t^{-1}[z_1] = \cdots = S_{i-1}^{-1}[z_1] = S_i^{-1}[z_1] = \cdots = S_{N-1}^{-1}[z_1]$
4: $j_i = 1$
Conditions: $S_t[i] = i$, $S_t^{-1}[z_1] \neq 1$, $S_t^{-1}[z_1] < t+1$ and $z_1 = S_t[S_t[1]+i]$
Key recovery relation: $\bar{K}[i] = 1 - \sigma_i$

Again, we assume $j_i = 1$. In the KSA, we know that $S_{i-1}[1] = S_t^{-1}[z_1] - i$ and $S_{i-1}[i] = i$. At the i-th stage, due to the swap we have $S_i[1] = i$ and $S_i[i] = S_t^{-1}[z_1] - i$. We assume these two values and $S_t^{-1}[z_1]$ are maintained until the end of the KSA. Now in the $PRGA$, initially

$i = 1$ and $j'_1 = S_{N-1}[1] = i$. So, we swap $S_{N-1}[1]$ and $S_{N-1}[i]$. Then, $z_1 = S'_1[S'_1[1] + S'_1[i]] = S'_1[S_t^{-1}[z_1] - i + i] = z_1$. Hence, the key recovery formula would be $\bar{K}[i] = 1 - \sigma_i$. The condition $S_t^{-1}[z_1] \neq 1$ is to filter the attack A_u13_3.

We classify the conditions as

$$\mathsf{C_1} : S_t[i] = i \qquad \text{and} \quad \mathsf{C_2} : S_t^{-1}[z_1] \neq 1, S_t^{-1}[z_1] < t+1 \qquad \text{and} \qquad \mathsf{C_3} : z_1 = S_t[S_t[1] + i]$$

We also classify the assumptions and the events and the key recovery bias as

$$\begin{cases} \mathsf{S_1} : S_t[i] = \cdots = S_{i-1}[i] \\ \mathsf{S_2} : S_i[1] = \cdots = S_{N-1}[1] \\ \mathsf{S_3} : S_t[1] = \cdots = S_{i-1}[1] \\ \mathsf{S_4} : S_i[i] = \cdots = S_{N-1}[i] \\ \mathsf{S_5} : S_t^{-1}[z_1] = \cdots = S_{N-1}^{-1}[z_1] \\ \mathsf{S_6} : \bar{K}[i] = j_i - \sigma_i \\ \mathsf{E_1} : j_i = 1 \\ \mathsf{B} : \bar{K}[i] = 1 - \sigma_i \end{cases}$$

Now, we compute the theoretical success probability of the attack. The goal is to estimate $\mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}]$. So, we compute

$$\begin{aligned} \mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}] &= \mathsf{Pr}[\mathsf{E_1}\mathsf{S_6}|\mathsf{C}] + \mathsf{Pr}[\mathsf{B}\neg\mathsf{S_3}|\mathsf{C}] \\ &= \mathsf{Pr}[\mathsf{E_1}|\mathsf{S_6}\mathsf{C}] \,.\, \mathsf{Pr}[\mathsf{S_6}|\mathsf{C}] + \mathsf{Pr}[\mathsf{B}|\neg\mathsf{S_6}\mathsf{C}] \,.\, (1 - \mathsf{Pr}[\mathsf{S_6}|\mathsf{C}]) \\ &\approx \mathsf{Pr}[\mathsf{E_1}|\mathsf{S_6}\mathsf{C}] \,.\, \mathsf{Pr}[\mathsf{S_6}|\mathsf{C}] + \left(\tfrac{1-\mathsf{Pr}[\mathsf{E_1}|\mathsf{S_6}\mathsf{C}]}{N-1}\right) \,.\, (1 - \mathsf{Pr}[\mathsf{S_6}|\mathsf{C}]) \\ &= \mathsf{Pr}(\mathsf{E_1}|\mathsf{S_6}\mathsf{C}) \,.\, \left(\tfrac{N\mathsf{Pr}[\mathsf{S_6}|\mathsf{C}]-1}{N-1}\right) + \left(\tfrac{1-P_B(i,t)}{N-1}\right) \end{aligned}$$

We then approximate $\mathsf{Pr}[\mathsf{S_6}|\mathsf{C}] \approx P_B(i,t)$ and we also have

$$\begin{aligned} \mathsf{Pr}[\mathsf{E_1}|\mathsf{S_6}\mathsf{C}] &\approx \mathsf{Pr}(\mathsf{E_1}|\mathsf{C}) \\ &= \mathsf{Pr}(\mathsf{C_1}\mathsf{C_2}|\mathsf{E_1}\mathsf{C_3}) \left(\tfrac{\mathsf{Pr}(\mathsf{E_1}|\mathsf{C_3})}{\mathsf{Pr}(\mathsf{C_1}\mathsf{C_2}|\mathsf{C_3})}\right) \\ &\approx \mathsf{Pr}(\mathsf{C_1}\mathsf{C_2}|\mathsf{E_1}\mathsf{C_3}) \\ &= \mathsf{Pr}(\mathsf{C_1}\mathsf{C_2}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{S_4}\mathsf{S_5}|\mathsf{E_1}\mathsf{C_3}) + \mathsf{Pr}(\mathsf{C_1}\mathsf{C_2}\neg(\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{S_4}\mathsf{S_5})|\mathsf{E_1}\mathsf{C_3}) \\ &\approx \mathsf{Pr}(\mathsf{C_1}\mathsf{C_2}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{S_4}\mathsf{S_5}|\mathsf{E_1}\mathsf{C_3}) + \tfrac{1}{N}(1 - \mathsf{Pr}(\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{S_4}\mathsf{S_5}|\mathsf{E_1}\mathsf{C_3})) \\ &\approx \mathsf{Pr}(\mathsf{C_1}\mathsf{C_2}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{S_4}\mathsf{S_5}|\mathsf{E_1}\mathsf{C_3}) + \tfrac{1}{N}\left(1 - P_A^1(i,t) \,.\, \left(\tfrac{N-1}{N}\right)^{N-i}\right) \end{aligned}$$

$$\begin{aligned} \mathsf{Pr}[\mathsf{C_1}\mathsf{C_2}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{S_4}\mathsf{S_5}|\mathsf{E_1}\mathsf{C_3}] &= \left(\frac{\mathsf{Pr}[\mathsf{C_1}\mathsf{C_2}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{S_4}\mathsf{S_5}\mathsf{E_1}|\mathsf{C_3}]}{\mathsf{Pr}[\mathsf{E_1}|\mathsf{C_3}]}\right) \\ &= \mathsf{Pr}[\mathsf{C_3}|\mathsf{C_1}\mathsf{C_2}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{S_4}\mathsf{S_5}\mathsf{E_1}] \,.\, \left(\frac{\mathsf{Pr}[\mathsf{C_1}\mathsf{C_2}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{S_4}\mathsf{S_5}\mathsf{E_1}]}{\mathsf{Pr}[\mathsf{C_3}] \,.\, \mathsf{Pr}[\mathsf{E_1}|\mathsf{C_3}]}\right) \end{aligned}$$

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}] = \mathsf{Pr}(\mathsf{E_1}|\mathsf{S_6}\mathsf{C}) \,.\, \left(\frac{NP_B(i,t)-1}{N-1}\right) + \left(\frac{1-P_B(i,t)}{N-1}\right)$$

where

$$\mathsf{Pr}[\mathsf{E_1}|\mathsf{S_6}\mathsf{C}] \approx \mathsf{Pr}(\mathsf{C_1}\mathsf{C_2}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{S_4}\mathsf{S_5}|\mathsf{E_1}\mathsf{C_3}) + \frac{1}{N}\left(1 - P_A^3(i,t) \,.\, \left(\frac{N-3}{N}\right)^{N-i-1}\right)$$

$$\begin{aligned}\Pr[C_1C_2S_1S_2S_3S_4S_5|E_1C_3] &= \left(\frac{\Pr[C_1C_2S_1S_2S_3S_4S_5E_1|C_3]}{\Pr[E_1|C_3]}\right)\\ &= \Pr[C_3|C_1C_2S_1S_2S_3S_4S_5E_1] \;.\; \left(\frac{\Pr[C_1C_2S_1S_2S_3S_4S_5E_1]}{\Pr[C_3] \;.\; \Pr[E_1|C_3]}\right)\end{aligned}$$

$\Pr[C_3]$ is uniformly distributed in this case and we also have

$$\Pr[S_t[i] = i] = \left(\frac{N-1}{N}\right)^{t+1}$$

Finally,

$$\begin{aligned}\Pr[E_1|C_3] &= \Pr[C_3|E_1]\left(\frac{\Pr[E_1]}{\Pr[C_3]}\right) = \Pr[C_3|E_1]\\ &= \Pr[C_3|E_1C_1C_2] \;.\; \Pr[C_1C_2|E_1] + \Pr[C_3|E_1\overline{C_1C_2}] \;.\; \Pr[\overline{C_1C_2}|E_1]\\ &= \Pr[C_1C_2|E_1] + \frac{1}{N}\left(1 - \Pr[C_1C_2|E_1]\right)\\ &= \left(1 - \frac{1}{N}\right)\Pr[C_1C_2|E_1] + \frac{1}{N}\end{aligned}$$

This leads to

$$\Pr[C_1C_2S_1S_2S_3S_4S_5|E_1C_3] = \frac{\left(\frac{N-1}{N}\right)^{t+1}\left(\frac{t}{N}\right)\left(\frac{N-3}{N}\right)^{N-1-i}P_A^3(i,t)}{\left(1-\frac{1}{N}\right)\left(\frac{N-1}{N}\right)^{t+1}\left(\frac{t}{N}\right) + \frac{1}{N}}$$

Therefore, overall we have

$$\begin{aligned}\Pr[B|C_1C_2C_3] &= \left(\frac{NP_B(i,t)-1}{N-1}\right)\\ &\quad\cdot\left[\frac{\left(\frac{N-1}{N}\right)^{t+1}\left(\frac{t}{N}\right)\left(\frac{N-3}{N}\right)^{N-1-i}}{\left(1-\frac{1}{N}\right)\left(\frac{N-1}{N}\right)^{t+1}\left(\frac{t}{N}\right)+\frac{1}{N}} \;.\; P_A^3(i,t) + \frac{1}{N}\left(1 - P_A^3(i,t)\left(\frac{N-3}{N}\right)^{N-i-1}\right)\right]\\ &\quad+\left(\frac{1-P_B(i,t)}{N-1}\right)\end{aligned}$$

6.2.11 The A_s3 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.11.

In the PRGA, at the first iteration $i = 1$ and $j'_1 = S_{N-1}[1] = \alpha$, then we swap $S_{N-1}[1]$ and $S_{N-1}[\alpha]$. Assume that $S'_1[2] = \beta$. At the next stage, $i = 2$ and $j'_2 = S'_1[2] + \alpha = \alpha + \beta$. Then, swap $S'_1[2] = \beta$ and $S'_1[\alpha+\beta]$. By one of the conditions, we have $S_t[2] + S_t[S_t[2] + S_t[1]] = i$. Therefore, we can write $\beta + S[\beta + \alpha] = i$. So, $S[\alpha + \beta] = i - \beta$. By the KSA, we have $S_i[i] = S_{i-1}[j_i]$ and $j_i = S_t^{-1}[z_2]$, so $S_i[i] = z_2$. If we look at how z_2 is generated, we have $z_2 = S'_2[S'_2[i] + S'_2[j'_2]] = S'_2[S'_2[2] + S'_2[\alpha+\beta]] = S'_2[i - \beta + \beta] = S'_2[i] = S_i[i] = z_2$. Using the same formulas as the previous attacks, we get $\bar{K}[i] = S_t^{-1}[z_2] - \sigma_i$. The condition $S_t[1] \neq 2$ prevents the value of $S_t[2]$ to be swapped in the first iteration of the PRGA. The condition $S_t[2] \neq 0$ prevents z_2 to be something except $S'_2[i]$, otherwise $z_2 = i - \beta$. The condition $S_t[1] + S_t[2] < t + 1$ makes its value not to be swapped in the next iterations of the KSA. We do not want the index of z_2 to be $1, 2$ nor $S_t[1]+S_t[2]$, because then, these values would be modified. So, we need to have $S_t^{-1}[z_2] \neq \{1, 2, S_t[1] + S_t[2]\}$.

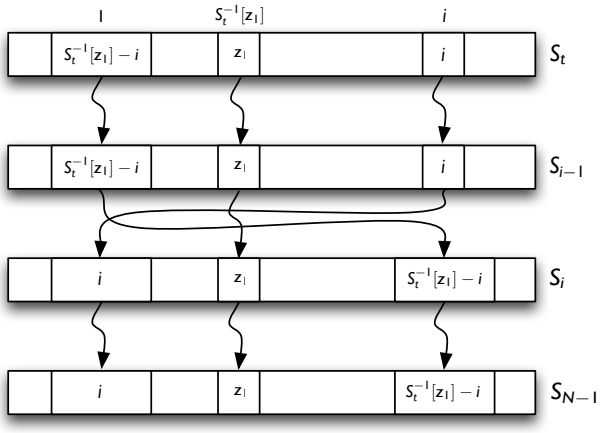


Figure 6.10: The RC4 state update in the A_u5_3 attack.

Algorithm 6.11 The A_s3 attack

Success Probability: Kor_3^4
Assumptions: (see Figure 6.11)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1]$
2: $S_t[2] = \cdots = S_{i-1}[2] = S_i[2] = \cdots = S_{N-1}[2]$
3: $S_t[S_t[1] + S_t[2]] = \cdots = S_{i-1}[S_{i-1}[1] + S_{i-1}[2]] = S_i[S_i[1] + S_i[2]] = \cdots = S_{N-1}[S_{N-1}[1] + S_{N-1}[2]]$
4: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i] = z_2$
Conditions: $S_t[1] \neq 2$, $S_t[2] \neq 0$, $S_t[2] + S_t[1] < t + 1$, $S_t[2] + S_t[S_t[2] + S_t[1]] = i$, $S_t^{-1}[z_2] \neq \{1, 2, S_t[1] + S_t[2]\}$, $S_t[1] + S_t[2] \neq \{1, 2\}$ and $(S_t^{-1}[z_2] < t + 1$ or $S_t^{-1}[z_2] > i - 1)$
Key recovery relation: $\bar{K}[i] = S_t^{-1}[z_2] - \sigma_i$

6.2.12 The A_s5_2 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.12.

Algorithm 6.12 The A_s5_2 attack

Success Probability: Kor_2^3
Assumptions: (see Figure 6.12)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1]$
2: $S_t[2] = \cdots = S_{i-1}[2] = S_i[2] = \cdots = S_{N-1}[2]$
3: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i]$
Conditions: $S_t[2] + S_t[1] = i$, $S_t^{-1}[S_t[1] - S_t[2]] \neq \{1, 2\}$, $(S_t^{-1}[S_t[1] - S_t[2]] < t + 1$ or $S_t^{-1}[S_t[1] - S_t[2]] > i - 1)$ and $z_2 = S_t[1]$
Key recovery relation: $\bar{K}[i] = S_t^{-1}[S_t[1] - S_t[2]] - \sigma_i$

In the PRGA, at the first stage $i = 1$ and $j_1' = S_{N-1}[1] = \alpha$. Then, we swap $S_{N-1}[1]$ and $S_{N-1}[\alpha]$.

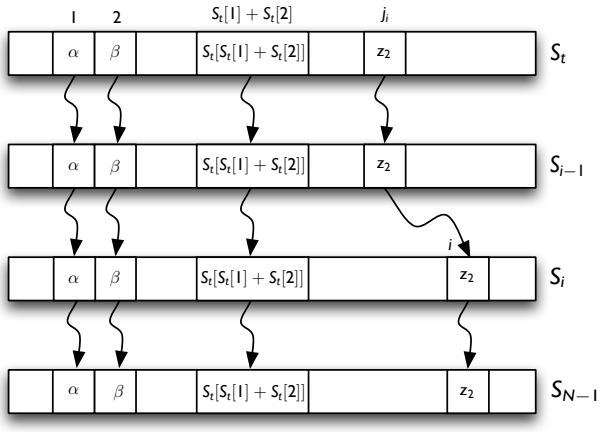


Figure 6.11: The RC4 state update in the A_s3 attack.

At the next iteration, $i = 2$ and $j'_2 = S'_1[2] + \alpha = \alpha + \beta = i$, where β is $S'_1[2]$ and from the conditions, we know that $\alpha + \beta = i$. Then, a swap is made between $S'_1[2]$ and $S'_1[i]$. Finally, $z_2 = S'_1[S'_1[2] + S'_1[i]]$. By the key recovery formula, we assume that $j_i = S_t^{-1}[S_t[1] - S_t[2]]$. Also, we know that $S_i[i] = S_{i-1}[j_i] = S_t[1] - S_t[2] = \alpha - \beta$. Therefore, $z_2 = S'_1[\alpha - \beta + \beta] = S'_1[\alpha] = \alpha = S_t[1]$. Hence, the key recovery formula is $K[i] = S_t^{-1}[S_t[1] - S_t[2]] - \sigma_i$. The condition $S_t^{-1}[S_t[1] - S_t[2]] \neq \{1, 2\}$ prevents j_i to be 1 or 2, so it prevents the swap of $S_{i-1}[1]$ and $S_{i-1}[2]$ in the i-th step of the KSA.

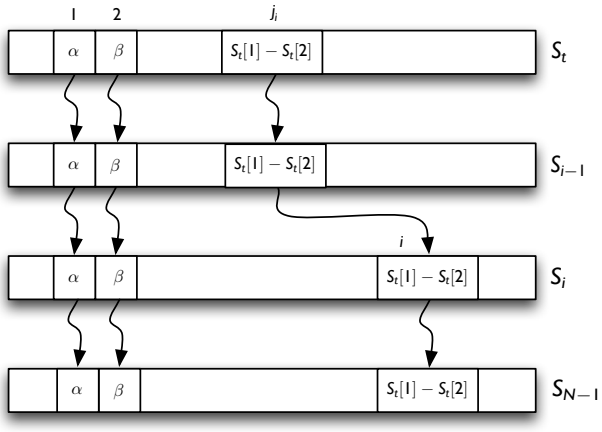


Figure 6.12: The RC4 state update in the A_s5_2 attack.

6.2.13 The A_s5_3 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.13.

Algorithm 6.13 The A_s5_3 attack

Success Probability: Kor_2^3
Assumptions: (see Figure 6.13)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1]$
2: $S_t[2] = \cdots = S_{i-1}[2] = S_i[2] = \cdots = S_{N-1}[2]$
3: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i]$
Conditions: $S_t[2] + S_t[1] = i$, $S_t^{-1}[z_2] \neq \{1, 2\}$, $(S_t^{-1}[2 - S_t[2]] < t + 1$ or $S_t^{-1}[2 - S_t[2]] > i - 1)$ and $z_2 = 2 - S_t[2]$
Key recovery relation: $\bar{K[i]} = S_t^{-1}[2 - S_t[2]] - \sigma_i$

In the PRGA, at the first stage $i = 1$ and $j'_1 = S_{N-1}[1] = \alpha$. Then, we swap $S_{N-1}[1]$ and $S_{N-1}[\alpha]$. At the next iteration, $i = 2$ and $j'_2 = S'_1[2] + \alpha = \alpha + \beta = i$, where β is $S'_1[2]$ and from the conditions, we know that $\alpha + \beta = i$. Then, a swap is made between $S'_1[2]$ and $S'_1[i]$. Finally, $z_2 = S'_1[S'_1[2] + S'_1[i]]$. By the key recovery formula, we assume that $j_i = S_t^{-1}[2 - S_t[2]]$. Also, we know that $S_i[i] = S_{i-1}[j_i] = 2 - S_t[2] = 2 - \beta$. Therefore, $z_2 = S'_1[2 - \beta + \beta] = S'_1[2] = 2 - S_t[2]$. Hence, the key recovery formula becomes $\bar{K[i]} = S_t^{-1}[2 - S_t[2]] - \sigma_i$. The condition $S_t^{-1}[z_2] \neq \{1, 2\}$ prevents j_i to be 1 or 2, so it prevents the swap of $S_{i-1}[1]$ and $S_{i-1}[2]$ in the i-th step of the KSA.

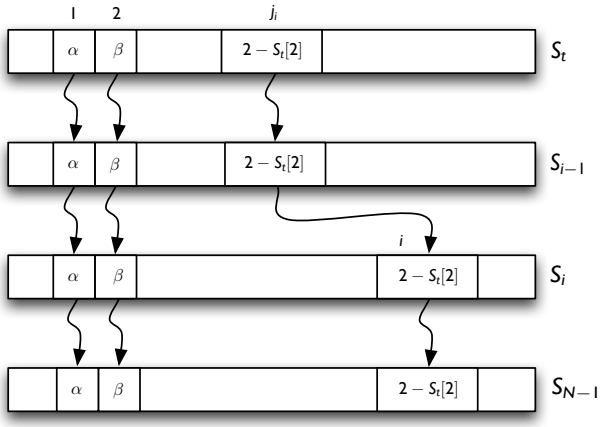


Figure 6.13: The RC4 state update in the A_s5_3 attack.

6.2.14 The A_4_s13 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.14.

This attack only works when $i = 4$. We also assume that $j_4 = S_t^{-1}[0]$. This assumption sets zero

Algorithm 6.14 The A_4_s13 attack

Success Probability: $P^4_{\mathsf{fixed}-j}$
Assumptions: (see Figure 6.14)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1]$
2: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i]$
3: $j_4 = S_t^{-1}[0]$
4: $i = 4$
Conditions: $S_t[1] = 2$, $S_t[4] \neq 0$, $(S_t^{-1}[0] < t+1$ or $S_t^{-1}[0] > i-1)$ and $z_2 = 0$
Key recovery relation: $\bar{K}[i] = S_t^{-1}[0] - \sigma_4$

into $S_4[4]$. In the $PRGA$, at the first iteration $i = 1$ and $j'_1 = S_{N-1}[1] = 2$. Then, we swap $S_{N-1}[1]$ and $S_{N-1}[2]$. At the next iteration, $i = 2$ and $j'_2 = S'_1[2] + 2 = 4$. Then, we swap $S'_1[2]$ and $S'_1[4]$. Finally, $z_2 = S'_2[S'_2[2] + S'_2[4]] = S'_2[2] = 0$. Hence, the formula for the key recovery becomes $S_t^{-1}[0] - \sigma_4$. We set the condition $S_t[4] \neq 0$ to differentiate this attack from the A_u15 attack.

We classify the conditions as

$$\begin{array}{llll} \mathsf{C_1} : S_t[1] = 2 & \text{and} & \mathsf{C_2} : S_t[4] \neq 0 \\ \mathsf{C_3} : (S_t^{-1}[0] < t+1 \text{ or } S_t^{-1}[0] > i-1) & \text{and} & \mathsf{C_4} : z_2 = 0 \end{array}$$

We also classify the assumptions, the events and the key recovery bias as

$$\begin{cases} \mathsf{S_1} : S_t[j_4] = \cdots = S_3[j_4] \\ \mathsf{S_2} : S_4[4] = \cdots = S_{N-1}[4] \\ \mathsf{S_3} : S_t[1] = \cdots = S_{N-1}[1] \\ \mathsf{S_4} : \bar{K}[i] = j_i - \sigma_i \\ \mathsf{E_1} : j_i = S_t^{-1}[0] \\ \mathsf{B} : \bar{K}[i] = S_t^{-1}[0] - \sigma_i \end{cases}$$

We compute the theoretical success probability of the attack. The goal is to estimate the probability $\mathsf{Pr}[\mathsf{B}|\mathsf{C_1}, \mathsf{C_2}, \mathsf{C_3}, \mathsf{C_4}]$. Deploying a similar approach to the one of the attack A_u15, we end up with

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C_1}\mathsf{C_2}\mathsf{C_3}\mathsf{C_4}] = \mathsf{Pr}(\mathsf{E_1}|\mathsf{C}) \ . \ \left(\frac{NP_B(i,t) - 1}{N-1}\right) + \left(\frac{1 - P_B(i,t)}{N-1}\right)$$

where

$$\mathsf{Pr}[\mathsf{E_1}|\mathsf{C}] \approx \mathsf{Pr}(\mathsf{C_1}\mathsf{C_2}\mathsf{C_3}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}|\mathsf{E_1}\mathsf{C_4}) + \frac{1}{N}\left(1 - P_A^2(i,t) \ . \ \left(\frac{N-2}{N}\right)^{N-i-1}\right)$$

$$\begin{aligned} \mathsf{Pr}[\mathsf{C_1}\mathsf{C_2}\mathsf{C_3}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}|\mathsf{E_1}\mathsf{C_4}] &= \left(\frac{\mathsf{Pr}[\mathsf{C_1}\mathsf{C_2}\mathsf{C_3}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{E_1}|\mathsf{C_4}]}{\mathsf{Pr}[\mathsf{E_1}|\mathsf{C_4}]}\right) \\ &= \mathsf{Pr}[\mathsf{C_4}|\mathsf{C_1}\mathsf{C_2}\mathsf{C_3}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{E_1}] \ . \ \left(\frac{\mathsf{Pr}[\mathsf{C_1}\mathsf{C_2}\mathsf{C_3}\mathsf{S_1}\mathsf{S_2}\mathsf{S_3}\mathsf{E_1}]}{\mathsf{Pr}[\mathsf{C_4}] \ . \ \mathsf{Pr}[\mathsf{E_1}|\mathsf{C_4}]}\right) \\ &= \frac{1}{2}\left(\frac{N-1}{N}\right)^{t+1}\left(\frac{N-2}{N}\right)^{N-1-i} \ . \ P_A^2(i,t) \end{aligned}$$

We know from Lemma 4.3 that $\mathsf{Pr}[\mathsf{C_4}] = \frac{2}{N}$ and we also have

$$\mathsf{Pr}[S_t[i] = i] = \left(\frac{N-1}{N}\right)^{t+1}$$

Therefore, overall we have

$$\begin{aligned}\mathsf{Pr}[\mathsf{B}|\mathsf{C_1C_2C_3}] &= \left(\frac{NP_B(i,t)-1}{N-1}\right) \\ &\cdot \left[\frac{1}{2}\left(\frac{N-1}{N}\right)^{t+1}\left(\frac{N-2}{N}\right)^{N-1-i} \cdot P_A^2(i,t) + \frac{1}{N}\left(1 - P_A^2(i,t)\left(\frac{N-2}{N}\right)^{N-i-1}\right)\right] \\ &+ \left(\frac{1-P_B(i,t)}{N-1}\right)\end{aligned}$$

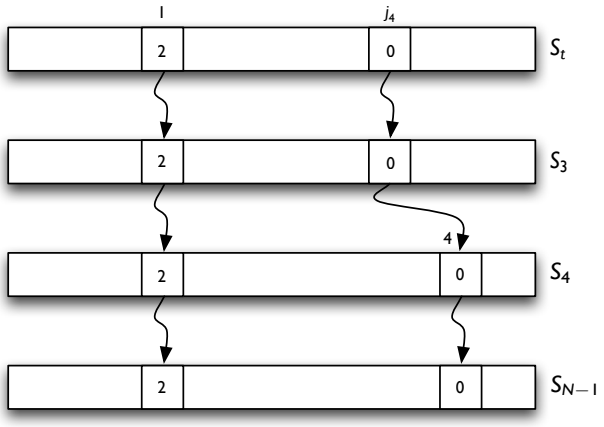


Figure 6.14: The RC4 state update in the A_4_s13 attack.

6.2.15 The A_4_u5_1 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.15.

This attack only works when $i = 4$. We also know that $j_i = S_t^{-1}[N-2]$. So, $S_i[i] = S_{i-1}[j_i] = N-2$. In the PRGA, at the first iteration $i = 1$ and $j'_1 = S_{N-1}[1] = 2$. Then, we swap $S_{N-1}[1]$ and $S_{N-1}[2]$. At the next iteration, $i = 2$ and $j'_2 = S'_1[2] + 2 = 4$. Then, we swap $S'_1[2]$ and $S'_1[4]$. Finally, $z_2 = S'_2[S'_2[2] + S'_2[4]] = S'_2[N-2+2] = S'_2[0]$. Hence, the formula for the key recovery becomes $S_t^{-1}[N-2] - \sigma_4$. We set the condition $z_2 \neq 0$ to differentiate this attack from the A_4_s13 attack.

6.2.16 The A_4_u5_2 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.16.

Algorithm 6.15 The A_4_u5_1 attack

Success Probability: Kor_2^3
Assumptions: (see Figure 6.15)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1] = 2$
2: $S_t[0] = \cdots = S_{i-1}[0] = S_i[0] = \cdots = S_{N-1}[0] = z_2$
3: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i]$
4: $i = 4$
Conditions: $S_t[1] = 2$, $z_2 \neq 0$, $z_2 \neq N-2$, $(S_t^{-1}[N-2] < t+1$ or $S_t^{-1}[N-2] > 3)$ and $z_2 = S_t[0]$
Key recovery relation: $\bar{K}[i] = S_t^{-1}[N-2] - \sigma_4$

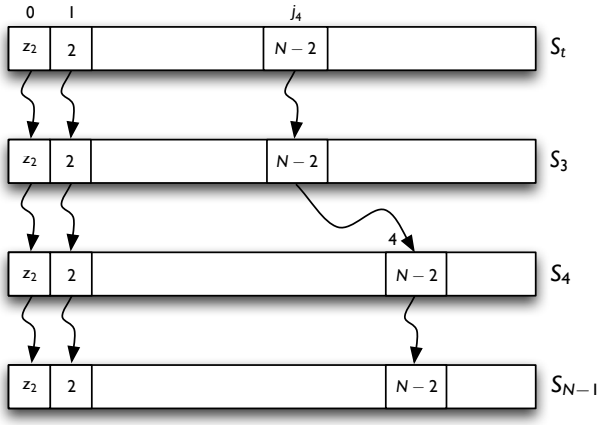


Figure 6.15: The RC4 state update in the A_4_u5_1 attack.

Algorithm 6.16 The A_4_u5_2 attack

Success Probability: Kor_2^3
Assumptions: (see Figure 6.16)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1] = 2$
2: $S_t[2] = \cdots = S_{i-1}[2] = S_i[2] = \cdots = S_{N-1}[2] = z_2$
3: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i]$
4: $i = 4$
Conditions: $S_t[1] = 2$, $z_2 \neq 0$, $(S_t^{-1}[N-1] < t+1$ or $S_t^{-1}[N-1] > 3)$ and $z_2 = S_t[2]$
Key recovery relation: $\bar{K}[i] = S_t^{-1}[N-1] - \sigma_4$

This attack only works when $i = 4$. We also know that $j_i = S_t^{-1}[N-1]$. So, $S_i[i] = S_{i-1}[j_i] = N-1$. In the PRGA, at the first iteration $i = 1$ and $j_1' = S_{N-1}[1] = 2$. Then, we swap $S_{N-1}[1]$ and $S_{N-1}[2]$. At the next iteration, $i = 2$ and $j_2' = S_1'[2] + 2 = 4$. Then, we swap $S_1'[2]$ and $S_1'[4]$. Finally, $z_2 = S_2'[S_2'[2] + S_2'[4]] = S_2'[N-1+2] = S_2'[1] = S_{N-1}[2] = S_t[2]$. Hence, the formula for the key recovery becomes $S_t^{-1}[N-1] - \sigma_4$. We set the condition $z_2 \neq 0$ to differentiate this attack from the A_4_s13 attack.

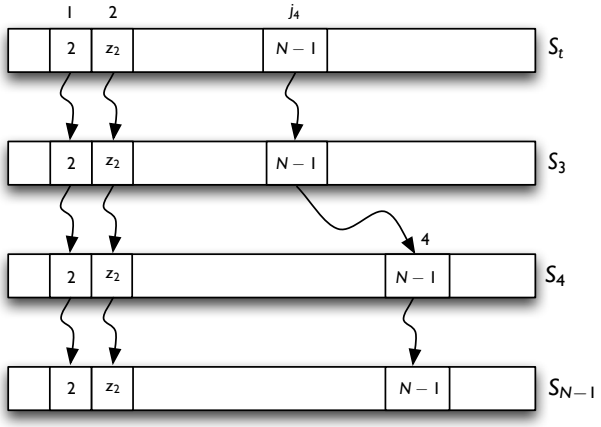


Figure 6.16: The RC4 state update in the A_4_u5_2 attack.

6.2.17 The A_neg_1 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.17.

Algorithm 6.17 The A_neg_1 attack

Success Probability: $P_{\text{neg}}(i,t)$
Assumptions: (see Figure 6.17)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1] = 2$
2: $S_t[2] = \cdots = S_{i-1}[2] = S_i[2] = \cdots = S_{N-1}[2] = 0$
Conditions: $S_t[2] = 0$, $S_t[1] = 2$ and $z_1 = 2$
Key recovery relation: $\bar{K}[i] = (1 - \sigma_i)$ or $\bar{K}[i] = (2 - \sigma_i)$

In the PRGA, at the first iteration $i = 1$ and $j'_1 = S_{N-1}[1] = 2$. Then, we swap $S_{N-1}[1]$ and $S_{N-1}[2]$. Finally, z_1 is computed as $z_1 = S'_1[S'_1[1] + S'_1[2]] = 2$. This means that $j_i \notin \{1, 2\}$, otherwise it moves $S_{i-1}[1]$ or $S_{i-1}[2]$ from their positions and so $z_1 = 2$ would not hold. Thus, we get $\bar{K}[i] \neq 1 - \sigma_i$ and $\bar{K}[i] \neq 2 - \sigma_i$.

At this stage, we compute the probability of these two negative biases. We define the following events and conditions.

$$\mathsf{E}_1 : j_i = 1 \text{ or } j_i = 2 \qquad \mathsf{B} : \bar{K}[i] = 1 - \sigma_i \text{ or } \bar{K}[i] = 2 - \sigma_i$$

$$\mathsf{C} : \begin{cases} \mathsf{C}_1 : S_t[2] = 0 \\ \mathsf{C}_2 : S_t[1] = 2 \\ \mathsf{C}_3 : z_1 = 2 \end{cases} \qquad \mathsf{S} : \begin{cases} \mathsf{S}_1 : S_t[1] = \cdots = S_{i-1}[1] \\ \mathsf{S}_2 : S_i[1] = \cdots = S_{N-1}[1] \\ \mathsf{S}_3 : S_t[2] = \cdots = S_{i-1}[2] \\ \mathsf{S}_4 : S_i[2] = \cdots = S_{N-1}[2] \\ \mathsf{S}_5 : \bar{K}[i] = j_i - \sigma_i \end{cases}$$

What we need is to compute $\mathsf{Pr}[\mathsf{B}|\mathsf{C}]$. It is computed as follows:

$$\begin{aligned}
\mathsf{Pr}[\mathsf{B}|\mathsf{C}] &= \mathsf{Pr}[\mathsf{E_1S_5}|\mathsf{C}] + \mathsf{Pr}[\mathsf{B}\neg\mathsf{S_5}|\mathsf{C}] \\
&= \mathsf{Pr}[\mathsf{E_1}|\mathsf{S_5C}]\mathsf{Pr}[\mathsf{S_5}|\mathsf{C}] + \mathsf{Pr}[\mathsf{B}\neg\mathsf{S_5}|\mathsf{C}] \\
&= \mathsf{Pr}[\mathsf{E_1}|\mathsf{S_5C}]\mathsf{Pr}[\mathsf{S_5}|\mathsf{C}] + \mathsf{Pr}[\mathsf{B}|\neg\mathsf{S_5C}]\,(1 - \mathsf{Pr}[\mathsf{S_5}|\mathsf{C}]) \\
&\approx \mathsf{Pr}[\mathsf{E_1}|\mathsf{S_5C}]\mathsf{Pr}[\mathsf{S_5}|\mathsf{C}] + \left(\tfrac{1-\mathsf{Pr}[\mathsf{E_1}|\mathsf{S_5C}]}{N-1}\right)(1 - \mathsf{Pr}[\mathsf{S_5}|\mathsf{C}]) \\
&= \mathsf{Pr}(\mathsf{E_1}|\mathsf{S_5C})\left(\tfrac{N\mathsf{Pr}[\mathsf{S_5}|\mathsf{C}]-1}{N-1}\right) + \left(\tfrac{1}{N-1}\right)(1 - \mathsf{Pr}[\mathsf{S_5}|\mathsf{C}])
\end{aligned}$$

We know that $\mathsf{Pr}[\mathsf{S_5}|\mathsf{C}] \approx P_B(i,t)$, so we just need to compute $\mathsf{Pr}[\mathsf{E_1}|\mathsf{S_5C}]$. Our approach is as follows:

$$\mathsf{Pr}[\mathsf{E_1}|\mathsf{S_5C}] \approx \mathsf{Pr}[\mathsf{E_1}|\mathsf{C}] = \mathsf{Pr}[\mathsf{C_3}|\mathsf{E_1C_1C_2}]\ .\ \left(\frac{\mathsf{Pr}[\mathsf{E_1}|\mathsf{C_1C_2}]}{\mathsf{Pr}[\mathsf{C_3}|\mathsf{C_1C_2}]}\right) \approx 0$$

So, overall, we have

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C}] = \left(\frac{1 - P_B(i,t)}{N-1}\right)$$

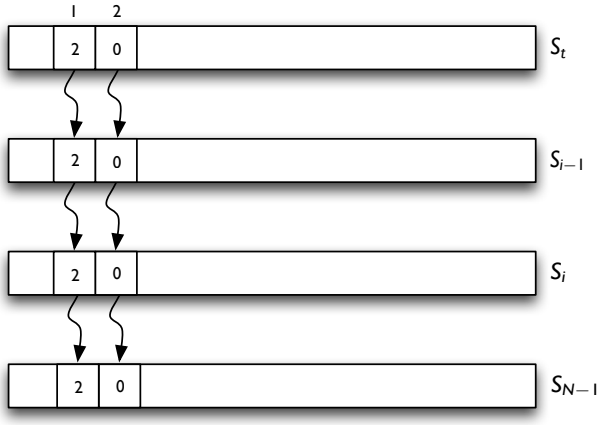


Figure 6.17: The RC4 state update in the A_neg_1 attack.

6.2.18 The A_neg_2 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.18.

Algorithm 6.18 The A_neg_2 attack

Success Probability: $P_{\text{neg}}(i,t)$
Assumptions: (see Figure 6.18)
1: $S_t[2] = \cdots = S_{i-1}[2] = S_i[2] = \cdots = S_{N-1}[2] = 0$
Conditions: $S_t[2] = 0$, $S_t[1] \neq 2$ and $z_2 = 0$
Key recovery relation: $K[i] = (2 - \sigma_i)$

In the PRGA, at the first iteration $i = 1$ and $j'_1 = S_{N-1}[1] = \alpha$. Then, we swap $S_{N-1}[1]$ and $S_{N-1}[\alpha]$. In the next iteration, $i = 2$ and $j'_2 = S'_1[2] + \alpha = \alpha$. Then, we swap $S'_1[2]$ and $S'_1[\alpha]$. Consequently, $z_2 = S'_2[S'_2[2] + S'_2[\alpha]] = S'_2[\alpha] = 0$. Similarly to the previous negative attacks, if $j_i = 2$, then $S_{i-1}[2]$ would be moved at the i-th step of the PRGA. To differentiate between this attack and the previous one, we set $S_t[1] \neq 2$. Finally, the filtering formula for the key would be $\bar{K[i]} = (2 - \sigma_i)$.

We define the following events and conditions.

$$\mathsf{E_1} : j_i = 2 \qquad \mathsf{B} : \bar{K}[i] = 2 - \sigma_i$$

$$\mathsf{C} : \begin{cases} \mathsf{C_1} : S_t[2] = 0 \\ \mathsf{C_2} = S_t[1] \neq 2 \\ \mathsf{C_3} : z_2 = 0 \end{cases} \qquad \mathsf{S} : \begin{cases} \mathsf{S_1} : S_t[2] = \cdots = S_{i-1}[2] \\ \mathsf{S_2} : S_i[2] = \cdots = S_{N-1}[2] \\ \mathsf{S_3} : \bar{K}[i] = j_i - \sigma_i \end{cases}$$

What we need is to compute $\mathsf{Pr}[\mathsf{B}|\mathsf{C}]$. It is computed as follows:

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C}] \approx \mathsf{Pr}(\mathsf{E_1}|\mathsf{S_3C}) \left(\frac{N\mathsf{Pr}[\mathsf{S_3}|\mathsf{C}] - 1}{N - 1} \right) + \left(\frac{1}{N - 1} \right) (1 - \mathsf{Pr}[\mathsf{S_3}|\mathsf{C}])$$

We know that $\mathsf{Pr}[\mathsf{S_3}|\mathsf{C}] \approx P_B(i, t)$, so we just need to compute $\mathsf{Pr}[\mathsf{E_1}|\mathsf{S_3C}]$. Our approach is as follows:

$$\mathsf{Pr}[\mathsf{E_1}|\mathsf{S_3C}] \approx \mathsf{Pr}[\mathsf{E_1}|\mathsf{C}] = \mathsf{Pr}[\mathsf{C_3}|\mathsf{E_1C_1C_2}] \; . \; \left(\frac{\mathsf{Pr}[\mathsf{E_1}|\mathsf{C_1C_2}]}{\mathsf{Pr}[\mathsf{C_3}|\mathsf{C_1C_2}]} \right) \approx 0$$

So, overall, we have

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C}] = \left(\frac{1 - P_B(i, t)}{N - 1} \right)$$

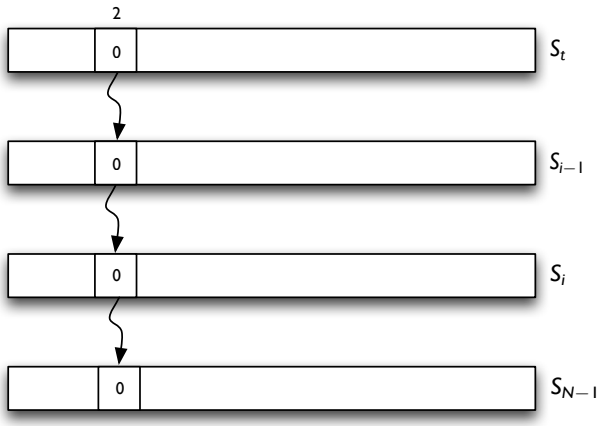


Figure 6.18: The RC4 state update in the A_neg_2 attack.

Algorithm 6.19 The A_neg_3 attack

Success Probability: $P_{\text{neg}}(i,t)$
Assumptions: (see Figure 6.19)
1: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1] = 1$
2: $S_t[2] = \cdots = S_{i-1}[2] = S_i[2] = \cdots = S_{N-1}[2] = z_1$
Conditions: $S_t[1] = 1$ and $z_1 = S_t[2]$
Key recovery relation: $\bar{K}[i] = (1 - \sigma_i)$ or $\bar{K}[i] = (2 - \sigma_i)$

6.2.19 The A_neg_3 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.19.

In the PRGA, at the first iteration $i = 1$ and $j'_1 = S_{N-1}[1] = 1$. After the swap, no element would be modified. Consequently, $z_1 = S'_1[S'_1[1] + S'_1[1]] = S'_1[2]$. Similarly to the previous negative attacks, if $j_i = 1$ or $j_i = 2$, then $S_{i-1}[1]$ or $S_{i-1}[2]$ would be moved at the i-th step of the PRGA. Finally, the filtering formula for the key would be $\bar{K}[i] = (1 - \sigma_i)$ or $\bar{K}[i] = (2 - \sigma_i)$ with a very low probability.

At this stage, we compute the probability of these two negative biases. We define the following events and conditions.

$$\mathsf{E_1} : j_i = 1 \text{ or } j_i = 2 \qquad \mathsf{B} : \bar{K}[i] = 1 - \sigma_i \text{ or } \bar{K}[i] = 1 - \sigma_i$$

$$\mathsf{C} : \begin{cases} \mathsf{C_1} : S_t[1] = 1 \\ \mathsf{C_2} : z_1 = S_t[2] \end{cases} \qquad \mathsf{S} : \begin{cases} \mathsf{S_1} : S_t[1] = \cdots = S_{i-1}[1] \\ \mathsf{S_2} : S_i[1] = \cdots = S_{N-1}[1] \\ \mathsf{S_3} : S_t[2] = \cdots = S_{i-1}[2] \\ \mathsf{S_4} : S_i[2] = \cdots = S_{N-1}[2] \\ \mathsf{S_5} : \bar{K}[i] = j_i - \sigma_i \end{cases}$$

What we need is to compute $\mathsf{Pr}[\mathsf{B}|\mathsf{C}]$. It is computed as follows:

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C}] \approx \mathsf{Pr}(\mathsf{E_1}|\mathsf{S_5C}) \left(\frac{N\mathsf{Pr}[\mathsf{S_5}|\mathsf{C}] - 1}{N - 1} \right) + \left(\frac{1}{N - 1} \right) (1 - \mathsf{Pr}[\mathsf{S_5}|\mathsf{C}])$$

We know that $\mathsf{Pr}[\mathsf{S_5}|\mathsf{C}] \approx P_B(i,t)$, so we just need to compute $\mathsf{Pr}[\mathsf{E_1}|\mathsf{S_5C}]$. Our approach is as follows:

$$\mathsf{Pr}[\mathsf{E_1}|\mathsf{S_5C}] \approx \mathsf{Pr}[\mathsf{E_1}|\mathsf{C}] = \mathsf{Pr}[\mathsf{C_2}|\mathsf{E_1C_1}] \; . \; \left(\frac{\mathsf{Pr}[\mathsf{E_1}|\mathsf{C_1}]}{\mathsf{Pr}[\mathsf{C_2}|\mathsf{C_1}]} \right) \approx 0$$

So, overall, we have

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C}] = \left(\frac{1 - P_B(i,t)}{N - 1} \right)$$

6.2.20 The A_neg_4 Attack

The conditions for this attack to succeed, the assumptions we make and the equation for the key recovery attack are represented in Algorithm 6.20.

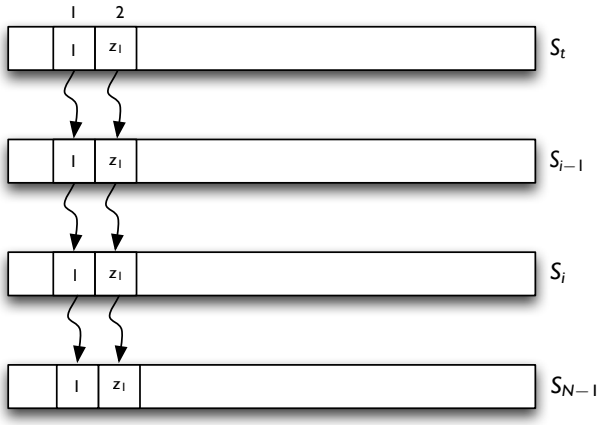


Figure 6.19: The RC4 state update in the A_neg_3 attack.

Algorithm 6.20 The A_neg_4 attack

Success Probability: $P_{\text{neg}}(i,t)$
Assumptions: (see Figure 6.20)
1: $S_t[0] = \cdots = S_{i-1}[0] = S_i[0] = \cdots = S_{N-1}[0] = 1$
2: $S_t[1] = \cdots = S_{i-1}[1] = S_i[1] = \cdots = S_{N-1}[1] = 0$
Conditions: $S_t[1] = 0$, $S_t[0] = 1$ and $z_1 = 1$
Key recovery relation: $\bar{K}[i] = (-\sigma_i)$ or $\bar{K}[i] = (1 - \sigma_i)$

In the PRGA, at the first iteration $i = 1$ and $j'_1 = S_{N-1}[1] = 0$. Then, $S_{N-1}[1]$ and $S_{N-1}[0]$ are swapped. Consequently, $z_1 = S'_1[S'_1[1] + S'_1[0]] = 1$. Similarly to the previous negative attacks, if $j_i = 0$ or $j_i = 1$, then $S_{i-1}[0]$ or $S_{i-1}[1]$ would be moved at the i-th step of the PRGA. Finally, the filtering formula for the key would be $\bar{K}[i] = (-\sigma_i)$ or $\bar{K}[i] = (1 - \sigma_i)$ which occurs with a low probability.

We compute the probability of these two negative biases. We define the following events and conditions.

$$\mathsf{E_1} : j_i = 0 \ \text{ or } \ j_i = 1 \qquad\qquad \mathsf{B} : \bar{K}[i] = -\sigma_i \ \text{ or } \ \bar{K}[i] = 1 - \sigma_i$$

$$C : \begin{cases} \mathsf{C_1} : S_t[0] = 1 \\ \mathsf{C_2} : S_t[1] = 0 \\ \mathsf{C_3} : z_1 = 1 \end{cases} \qquad\qquad \mathsf{S} : \begin{cases} \mathsf{S_1} : S_t[0] = \cdots = S_{i-1}[0] \\ \mathsf{S_2} : S_i[0] = \cdots = S_{N-1}[0] \\ \mathsf{S_3} : S_t[1] = \cdots = S_{i-1}[1] \\ \mathsf{S_4} : S_i[1] = \cdots = S_{N-1}[1] \\ \mathsf{S_5} : \bar{K}[i] = j_i - \sigma_i \end{cases}$$

What we need is to compute $\mathsf{Pr}[\mathsf{B}|\mathsf{C}]$. It is computed as follows:

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C}] \approx \mathsf{Pr}(\mathsf{E_1}|\mathsf{S_5C}) \left(\frac{N\mathsf{Pr}[\mathsf{S_5}|\mathsf{C}] - 1}{N - 1} \right) + \left(\frac{1}{N-1} \right) (1 - \mathsf{Pr}[\mathsf{S_5}|\mathsf{C}])$$

We know that $\mathsf{Pr}[\mathsf{S_5}|\mathsf{C}] \approx P_B(i,t)$, so we just need to compute $\mathsf{Pr}[\mathsf{E_1}|\mathsf{S_5C}]$. Our approach is as

follows:

$$\mathsf{Pr}[\mathsf{E}_1|\mathsf{S}_3\mathsf{C}] \approx \mathsf{Pr}[\mathsf{E}_1|\mathsf{C}] = \mathsf{Pr}[\mathsf{C}_3|\mathsf{E}_1\mathsf{C}_1\mathsf{C}_2] \; . \; \left(\frac{\mathsf{Pr}[\mathsf{E}_1|\mathsf{C}_1\mathsf{C}_2]}{\mathsf{Pr}[\mathsf{C}_3|\mathsf{C}_1\mathsf{C}_2]}\right) \approx 0$$

So, overall, we have

$$\mathsf{Pr}[\mathsf{B}|\mathsf{C}] = \left(\frac{1 - P_B(i,t)}{N-1}\right)$$

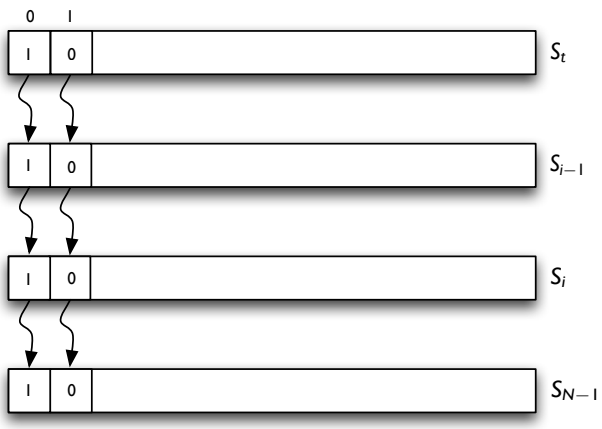


Figure 6.20: The RC4 state update in the A_neg_4 attack.

6.3 Computation of the Biases

The biases were computed using the following formulas:

$$
\begin{aligned}
\mathsf{Kor}_c^b(i,t) &= R_c^b(i,t) \otimes P_B(i,t) \\
P_{\text{neg}}(i,t) &= \left(\tfrac{1-P_B(i,t)}{N-1}\right) \\
P_{\mathsf{SVV10}}(t) &= P_{\mathsf{db2}} \otimes P_A^1(16,t) \otimes P_B(16,t) \\
P^1_{\mathsf{fixed}-j} &= \left(\tfrac{NP_B(i,t)-1}{N-1}\right) \cdot \left[\tfrac{1}{2} P_A^1(i,t) \left(\tfrac{N-1}{N}\right)^{N-i} + \tfrac{1}{N}\left(1 - P_A^1(i,t)\left(\tfrac{N-1}{N}\right)^{N-i}\right)\right] \\
&\quad + \left(\tfrac{1-P_B(i,t)}{N-1}\right) \\
P^2_{\mathsf{fixed}-j} &= \tfrac{1}{N}\left(\tfrac{NP_B(i,t)-1}{N-1}\right) \cdot \left[\tfrac{1}{\xi} P_A^2(i,t) \left(\tfrac{N}{N-1}\right)^{t-2} \left(\tfrac{N-2}{N}\right)^{N-1-i} + \left(1 - P_A^2(i,t)\left(\tfrac{N-2}{N}\right)^{N-i-1}\right)\right] \\
&\quad + \left(\tfrac{1-P_B(i,t)}{N-1}\right) \\
P^3_{\mathsf{fixed}-j} &= \left(\tfrac{NP_B(i,t)-1}{N-1}\right) \cdot \left[\left(\tfrac{N-1}{N}\right)^{t+1} \left(\tfrac{N-2}{N}\right)^{N-1-i} \cdot P_A^2(i,t) + \tfrac{1}{N}\left(1 - P_A^2(i,t)\left(\tfrac{N-2}{N}\right)^{N-i-1}\right)\right] \\
&\quad + \left(\tfrac{1-P_B(i,t)}{N-1}\right) \\
P^4_{\mathsf{fixed}-j} &= \left(\tfrac{NP_B(i,t)-1}{N-1}\right) \cdot \left[\tfrac{1}{2}\left(\tfrac{N-1}{N}\right)^{t+1} \left(\tfrac{N-2}{N}\right)^{N-1-i} \cdot P_A^2(i,t) + \tfrac{1}{N}\left(1 - P_A^2(i,t)\left(\tfrac{N-2}{N}\right)^{N-i-1}\right)\right] \\
&\quad + \left(\tfrac{1-P_B(i,t)}{N-1}\right) \\
P^5_{\mathsf{fixed}-j} &= \left(\tfrac{NP_B(i,t)-1}{N-1}\right) \cdot \left[\tfrac{\left(\frac{N-1}{N}\right)^{t+1}\left(\frac{t}{N}\right)\left(\frac{N-3}{N}\right)^{N-1-i}}{\left(1-\frac{1}{N}\right)\left(\frac{N-1}{N}\right)^{t+1}\left(\frac{t}{N}\right)+\frac{1}{N}} \cdot P_A^3(i,t) + \tfrac{1}{N}\left(1 - P_A^3(i,t)\left(\tfrac{N-3}{N}\right)^{N-i-1}\right)\right] \\
&\quad + \left(\tfrac{1-P_B(i,t)}{N-1}\right)
\end{aligned}
$$

where $P_{\mathsf{db2}} = \frac{9.444}{N}$ and $\xi = \frac{1}{N}\left[\left(\frac{N-1}{N}\right)^N \left(1 - \frac{1}{N} + \frac{1}{N^2}\right) + \frac{1}{N^2} + 1\right]$.

$$
\begin{aligned}
P_A^b(i,t) &= \left(\tfrac{N-b}{N}\right)^{i-t-1} \\
P_B(i,t) &= \textstyle\prod_{k=0}^{i-t-1}\left(\frac{N-k}{N}\right) + \frac{1}{N}\left(1 - \prod_{k=0}^{i-t-1}\left(\frac{N-k}{N}\right)\right) \\
R_c^b(i,t) &= r_c(i) P_A^b(i,t) + \tfrac{1}{N}(1 - r_c(i) P_A^b(i,t)) \\
r_1(i) &= \left(\tfrac{N-2}{N}\right)^{N-i-1} \\
r_2(i) &= \left(\tfrac{N-3}{N}\right)^{N-i-1} \\
r_3(i) &= \left(\tfrac{N-4}{N}\right)^{N-i-1}
\end{aligned}
$$

These formulas are new. The biases were originally provided with probabilities for $t = -1$.

Table 6.1: The Conditional Biases for RC4, exploitable against WEP and WPA

row	reference	$\bar{f}$	$\bar{g}$	p
i	A_u15	$2-\sigma_i$	$S_t[i]=0,\ z_2=0$	$P^1_{\mathsf{fixed}-j}$
i	A_s13	$S_t^{-1}[0]-\sigma_i$	$S_t[1]=i,\ (S_t^{-1}[0]<t+1$ or $S_t^{-1}[0]>i-1),\ z_1=i$	Kor_1^2
i	A_u13_1	$S_t^{-1}[z_1]-\sigma_i$	$S_t[1]=i,\ (S_t^{-1}[z_1]<t+1$ or $S_t^{-1}[z_1]>i-1),\ z_1=1-i$	Kor_1^2
i	A_u13_2	$1-\sigma_i$	$S_t[i]=i,\ S_t[1]=0,\ z_1=i$	$P^3_{\mathsf{fixed}-j}$
i	A_u13_3	$1-\sigma_i$	$S_t[i]=i,\ S_t[1]=1-i,\ z_1=1-i$	$P^3_{\mathsf{fixed}-j}$
i	A_s5_1	$S_t^{-1}[z_1]-\sigma_i$	$S_t[1]<t+1,\ S_t[1]+S_t[S_t[1]]=i,\ z_1\neq\{S_t[1],S_t[S_t[1]]\},\ (S_t^{-1}[z_1]<t+1$ or $S_t^{-1}[z_1]>i-1)$	Kor_2^3
i	A_s5_2	$S_t^{-1}[S_t[1]-S_t[2]]-\sigma_i$	$S_t[2]+S_t[1]=i,\ S_t^{-1}[S_t[1]-S_t[2]]\neq\{1,2\},\ (S_t^{-1}[S_t[1]-S_t[2]]<t+1$ or $S_t^{-1}[S_t[1]-S_t[2]]>i-1),\ z_2=S_t[1]$	Kor_2^3
i	A_s5_3	$S_t^{-1}[z_2]-\sigma_i$	$S_t[2]+S_t[1]=i,\ S_t^{-1}[z_2]\neq\{1,2\},\ (S_t^{-1}[z_2]<t+1$ or $S_t^{-1}[z_2]>i-1),\ z_2=2-S_t[2]$	Kor_2^3
i	A_u5_1	$S_t^{-1}[S_t^{-1}[z_1]-i]-\sigma_i$	$S_t[1]=i,\ S_t^{-1}[z_1]<t+1,\ S_t^{-1}[S_t^{-1}[z_1]-i]\neq 1,\ (S_t^{-1}[S_t^{-1}[z_1]-i]<t+1$ or $S_t^{-1}[S_t^{-1}[z_1]-i]>i-1),\ z_1\neq\{i,1-i,S_t^{-1}[z_1]-i\},\ S_t^{-1}[z_1]\neq 2i$	Kor_2^3
i	A_u5_2	$1-\sigma_i$	$S_t[i]=1,\ z_1=S_t[2]$	$P^2_{\mathsf{fixed}-j}$
i	A_u5_3	$1-\sigma_i$	$S_t[i]=i,\ S_t^{-1}[z_1]\neq 1,\ S_t^{-1}[z_1]<t+1,\ z_1=S_t[S_t[1]+i]$	$P^5_{\mathsf{fixed}-j}$
i	A_s3	$S_t^{-1}[z_2]-\sigma_i$	$S_t[1]\neq 2,\ S_t[2]\neq 0,\ S_t[2]+S_t[1]<t+1,\ S_t[2]+S_t[S_t[2]+S_t[1]]=i,\ S_t^{-1}[z_2]\neq\{1,2,S_t[1]+S_t[2]\},\ S_t[1]+S_t[2]\neq\{1,2\},\ (S_t^{-1}[z_2]<t+1$ or $S_t^{-1}[z_2]>i-1)$	Kor_3^4
4	A_4_s13	$S_t^{-1}[0]-\sigma_4$	$S_t[1]=2,\ S_t[4]\neq 0,\ (S_t^{-1}[0]<t+1$ or $S_t^{-1}[0]>i-1),\ z_2=0$	$P^4_{\mathsf{fixed}-j}$
4	A_4_u5_1	$S_t^{-1}[N-2]-\sigma_4$	$S_t[1]=2,\ z_2\neq 0,\ z_2=S_t[0],\ z_2\neq N-2,\ (S_t^{-1}[N-2]<t+1$ or $S_t^{-1}[N-2]>3)$	Kor_2^3
4	A_4_u5_2	$S_t^{-1}[N-1]-\sigma_4$	$S_t[1]=2,\ z_2\neq 0,\ (S_t^{-1}[N-1]<t+1$ or $S_t^{-1}[N-1]>3),\ z_2=S_t[2]$	Kor_2^3
i	A_neg_1	$1-\sigma_i$ or $2-\sigma_i$	$S_t[2]=0,\ S_t[1]=2,\ z_1=2$	$P_{\mathrm{neg}}(i,t)$
i	A_neg_2	$2-\sigma_i$	$S_t[2]=0,\ S_t[1]\neq 2,\ z_2=0$	$P_{\mathrm{neg}}(i,t)$
i	A_neg_3	$1-\sigma_i$ or $2-\sigma_i$	$S_t[1]=1,\ z_1=S_t[2]$	$P_{\mathrm{neg}}(i,t)$
i	A_neg_4	$-\sigma_i$ or $1-\sigma_i$	$S_t[1]=0,\ S_t[0]=1,\ z_1=1$	$P_{\mathrm{neg}}(i,t)$
16	SVV_10	$S_t^{-1}[0]-\sigma_{16}$	$S_t^{-1}[0]<t+1$ or $S_t^{-1}[0]>15,\ z_{16}=-16,\ j_2\notin\{t+1,\ldots,15\}$	$P_{\mathsf{SVV10}}(t)$

Chapter 7

Cryptanalysis of the WPA Protocol

In this chapter, we introduce two types of attacks against the WPA protocol. The first is a key recovery attack and the second is a distinguishing attack. Firstly, based on several partial temporary key recovery attacks, we recover the full 128-bit temporary key by using 2^{42} packets. It works with the complexity of 2^{96}. Then, we describe a distinguisher for WPA of complexity 2^{42} and advantage 0.5 which uses 2^{42} packets. So far, this is the best attack against WPA. We believe that our analysis brings further insights to the security of RC4.

Note. We analyzed four unconditional biases in Chapter 5, i.e., the Klein-Improved correlation, the SVV_008, the SVV_009 and the Maitra-Paul-Improved correlation. The success probability of SVV_008 and SVV_009 are quite low. Hence, we avoid these two biases during our experiments. Furthermore, we found out that using the Maitra-Paul-Improved bias together with the Klein-Improved attack bring only a low extra success probability. Hence, during all our experiments on WPA and later on WEP, we also avoided the Maitra-Paul-Improved attack. Regarding the conditional attacks, we used all the Korek attacks together with the SVV_10 attack.

7.1 Useful Notations

We use a list of classes of biases from Table 5.1 and 6.1. More specifically, we use the rows $\mathsf{row}^{\mathsf{RC4}}_{i'|I_0}$ in those tables. Furthermore, there exists a prominent relation between the key bytes of RC4:

$$\bar{K}[i+16j] = \bar{K}[i] + j\bar{K}[15]$$

for $0 \le i \le 15$ and $j = 0, 1, 2$. This relation reveals that if $\bar{K}[15]$ is known, the biases for $\bar{K}[i]$ and $\bar{K}[i+16j]$ can be merged. Later, we recover $\bar{K}[15]$ before any other byte of the key.

We define $\mathsf{deduce}(I)$ to be the set of all i's such that we can compute $\bar{K}[i]$ using this property, based on the values of $\bar{K}$ with indices in I. For instance, $\mathsf{deduce}(0,1,2,5) = \{0,1,2,5\}$ and $\mathsf{deduce}(0,1,2,5,15) = \{0,1,2,5,15,16,17,18,21,31,32,33,34,37,\ldots\}$. Next, we transform the above table by removing some rows for the keys which can be deduced and by merging the rows leading to the same key byte. Namely, we define $\mathsf{row}_{i|I_0}$ as follows: if $i \in \mathsf{deduce}(I_0)$, the row has a single "bias" $\bar{f}_1(z, \mathsf{clue}) = \bar{K}[i]$ with probability $p_1 = 1$, as $\bar{K}[i]$ can be computed from the clue. Otherwise, the row is the concatenation of all $\mathsf{row}^{\mathsf{RC4}}_{i'|I_0}$ for i' such that $i \in \mathsf{deduce}(I_0 \cup \{i'\})$. For

instance, $\mathsf{row}_{2|\{0,1,2\}}$ has a single bias, $\mathsf{row}_{5|\{0,1,2\}} = \mathsf{row}^{\mathsf{RC4}}_{5|\{0,1,2\}}$, and

$$\mathsf{row}_{5|\{0,1,2,15\}} = \mathsf{row}^{\mathsf{RC4}}_{5|\{0,1,2,15\}} \| \mathsf{row}^{\mathsf{RC4}}_{21|\{0,1,2,15\}} \| \mathsf{row}^{\mathsf{RC4}}_{37|\{0,1,2,15\}}$$

In the "concatenation" above, we only update $\bar{f}_j$ from the $\mathsf{row}_{i'|I_0}$ so that it computes the $\bar{K}[i]$ instead of the $\bar{K}[i']$. Given two lists of byte indices I_0 and $I = (i_1, \ldots, i_{\#I})$, we construct a new table $\Pi(I|I_0)$ in which the list of rows is $\mathsf{row}_{i_1|I_0}$, $\mathsf{row}_{i_2|I_0,i_1}$, ..., $\mathsf{row}_{i_{\#I}|I_0,i_1,i_2,\ldots,i_{\#I-1}}$. For instance, $I_0 = \{0,1,2\}$ and I is a list of the key byte indices which are sequentially obtained using the biases. We assume that I_0 is a minimal set in the sense that there is no strictly smaller set with the same $\mathsf{deduce}(I_0)$. We further assume that I is a minimal set in the sense that there is no strictly smaller set with the same $I \cap I_0$ and $\mathsf{deduce}(I \cup I_0)$. For instance, $I = (2,3,13,14,15)$ is minimal for $I_0 = \{0,1,2\}$, but $I = (2,3,13,14,15,16)$ is not. We define $\nu = (\bar{K}[i])_{i \in I}$ which belongs to a set of size $N_\nu(I) = N^{\#I}$. Given $i \in I$, we let $d_i^{\Pi(I|I_0)}$ be the length of the row for $\bar{K}[i]$ in $\Pi(I|I_0)$. Given a tuple $(j_i)_{i \in I}$ such that $1 \leq j_i \leq d_i^{\Pi(I|I_0)}$ for all $i \in I$, by collecting the j_i-th bias of the row i, we obtain an agglomerated bias to compute ν from z and an I_0-clue clue. Note that for technical reasons, we may have to keep elements of I_0 in I. This is why we may have rows for $i \in I_0$ in $\Pi(I|I_0)$ with a single bias with probability 1. We let

$$k(I|I_0) = \prod_{i \in I} d_i^{\Pi(I|I_0)}$$

be the number of possible agglomerated biases. For convenience, we number the agglomerated biases with an index ℓ from 1 to $k(I|I_0)$, where each number defines a tuple $(j_i)_{i \in I}$. So, the ℓ-th bias is defined by $\nu = f_\ell(z, \mathsf{clue})$ with probability

$$p_\ell^{\Pi(I|I_0)} = \prod_{i \in I} p_{i,j_i}^{\Pi(I|I_0)}$$

where $p_{i,j}^{\Pi(I|I_0)}$ is the probability of the j-th bias in the row corresponding to $\bar{K}[i]$ in $\Pi(I|I_0)$.

We let $N_\mu(\Pi(I))$ be N raised to the power of the number of z_i bytes and I_0 bytes appearing in any of the biased equations from $\Pi(I)$. For example, $N_\mu(\Pi(3,13,14|0,1,2)) = N^8$, as biases for $\bar{K}[3]$ are based on z_3 and z_4 and biases for $\bar{K}[13]$ and $\bar{K}[14]$ are based on z_{13}, z_{14} and z_{15}. We further need the IV to compute the S_t. So, we have 8 bytes in total: z_i for $i \in \{3,4,13,14,15\}$ and the IV. Given a keystream z, we define $\mu = h^{\Pi(I)}(z, \mathsf{clue})$ as the vector of all z_i and clue bytes which are useful. We define $\nu = f_\ell^{\Pi(I)}(\mu)$.

For simplicity, we write Π, k, N_ν, N_μ, p_ℓ, h and f_ℓ when I and I_0 are made clear from context. That is, the range of h has size N_μ, and f_ℓ goes from a domain of N_μ elements to a range of N_ν elements.

Finally, this section describes a way to construct a list Π of k biases for the vector $\nu = (\bar{K}[i])_{i \in I}$ from $\mu = h(z, \mathsf{clue})$.

7.2 A Key Recovery attack on WPA

There are 8 bits of the TK that we call *weak*, because they have a simple relation with the bits of the PPK. These bits consist of the 7 most significant bits of the $\mathsf{TK}[0]$ and the least significant bit

of the $\mathsf{TK}[1]$. We define some statistical attacks using the following mappings:

$$z^m, \mathsf{IV}^m \xrightarrow{h} \mu \xrightarrow[\text{if } g_\ell(\mu)]{f_\ell} \nu \xrightarrow{\pi} x$$

Here, z^m is the m-th keystream using the IV^m and μ is some compressed information to compute ν. The ν is some RC4 key bytes which are useful in computing x. The x is some information about the TK which we want to recover using statistics. We define N_x as the number of possible values for x.

7.2.1 The First Attack: Recovering some Weak Bits of TK

We use $I_0 = \{0, 1, 2\}$ and $I = (2, 3, 13, 14)$. Given $\bar{K}[2]$, $\bar{K}[3]$, $\bar{K}[13]$ and $\bar{K}[14]$, the adversary can compute $K[3] = \bar{K}[3] - \bar{K}[2]$ and $K[14] = \bar{K}[14] - \bar{K}[13]$. According to Algorithm 2.2, we have

$$\begin{aligned} \mathsf{PPK}[5] &= K[15] \| K[14] \\ K[3] &= \mathsf{low8}((\mathsf{PPK}[5] \oplus (\mathsf{RotR1}(\mathsf{TK}[1] \| \mathsf{TK}[0])))) \end{aligned}$$

So, given $\nu = (\bar{K}[2], \bar{K}[3], \bar{K}[13], \bar{K}[14])$, the adversary can compute $x = \mathsf{high7}(\mathsf{TK}[0])$ by

$$\pi(\nu) = \mathsf{low7}((\bar{K}[3] - \bar{K}[2]) \oplus (\mathsf{RotR1}(\bar{K}[14] - \bar{K}[13])))$$

$N_\nu = 2^{32}$ is the total number of possible ν's and $N_x = 2^7$ is the total number of possible x's. We have $N_\mu = 2^{48}$, the total number of $\mu = h(z, \mathsf{IV})$.

We can recover the 7 weak bits as follows: for each candidate value x, each packet m, and each $\ell = 1, \ldots, k$ (corresponding to a tuple $(j_2, j_3, j_{13}, j_{14})$, if the agglomerated condition $g_\ell(h(z^m, \mathsf{IV}^m))$ holds, we define $\nu = f_\ell(h(z^m, \mathsf{IV}^m))$ as the value of the RC4 key bytes suggested by the bias ℓ on packet m, which is correct with probability p_ℓ. We let $x = \pi(\nu)$ be the suggested value of x computed as explained. We let $X_{x,m,\ell}$ be some magic coefficient a_ℓ (to be optimized later) if $\pi(f_\ell(h(z^m, \mathsf{IV}^m))) = x$ and 0 otherwise. We let

$$Y_x = \sum_{m=1}^{n} \sum_{\ell=1}^{k} X_{x,m,\ell}$$

where n is the total number of packets to be used. Clearly, the correct value for ν is suggested with probability p_ℓ and others are obtained randomly. We assume the incorrect ones are suggested with the same probability $\frac{1-p_\ell}{N_\nu - 1}$.

If x is not the correct value, it is not suggested for sure when ν is correct. Since π is balanced, this incorrect x has $\frac{N_\nu}{N_x}$ values ν belonging to the set of $N_\nu - 1$ incorrect ones. So, x is suggested with probability $\frac{N_\nu}{N_x} \times \frac{1-p_\ell}{N_\nu - 1}$. So, the $X_{x,m,\ell}$ for the incorrect x's are random variables with the expected values

$$a_\ell q_\ell N_\nu \frac{1 - p_\ell}{N_x(N_\nu - 1)}$$

if x is not the correct value.

If x is the correct value, it is suggested with probability p_ℓ for the correct ν and by π when ν is one of the $\frac{N_\nu - N_x}{N_x}$ (incorrect) preimages of x. That is, with overall probability $p_\ell + \frac{N_\nu - N_x}{N_x} \times \frac{1-p_\ell}{N_\nu - 1}$. So, the $X_{x,m,\ell}$ for the correct x are random variables with the expected value

$$a_\ell q_\ell N_\nu \frac{1 - p_\ell}{N_x(N_\nu - 1)} + a_\ell q_\ell \frac{N_\nu p_\ell - 1}{N_\nu - 1}$$

The difference between these two expected values is important. This is also the case for the difference of the variances. As every x is suggested with probability roughly $\frac{q_\ell}{N_x}$, we assume that the variance of a bad $X_{x,m,\ell}$ can be approximated by $\frac{q_\ell}{N_x}\left(1-\frac{q_\ell}{N_x}\right)a_\ell^2$. Let Δ be the operator making the difference between the distributions of a good x and a bad one. We have

$$\begin{aligned}
E(Y_{x\ \mathsf{bad}}) &= \frac{n}{N_x\left(1-\frac{1}{N_\nu}\right)}\sum_\ell a_\ell q_\ell(1-p_\ell)\\
E(Y_{x\ \mathsf{good}}) &= E(Y_{x\ \mathsf{bad}})+\Delta E(Y)\\
\Delta E(Y) &= \frac{n}{1-\frac{1}{N_\nu}}\sum_\ell a_\ell q_\ell\left(p_\ell-\frac{1}{N_\nu}\right)\\
V(Y_{x\ \mathsf{bad}}) &\approx n\sum_\ell a_\ell^2\frac{q_\ell}{N_x}\left(1-\frac{q_\ell}{N_x}\right)\\
V(Y_{x\ \mathsf{good}}) &= V(Y_{x\ \mathsf{bad}})+\Delta V(Y)\\
\Delta V(Y) &\approx \frac{n}{1-\frac{1}{N_\nu}}\sum_\ell a_\ell^2 q_\ell\left(p_\ell-\frac{1}{N_\nu}\right)
\end{aligned}$$

where $E(Y_{x\ \mathsf{bad}})$ and $V(Y_{x\ \mathsf{bad}})$ denote the expected value and the variance of a Y_x variable for any bad x respectively. Here, we removed the subscript x of Y_x in $\Delta E(Y)$ as this does not depend on a specific value for x. Let λ be such that $\Delta E(Y)=\lambda\sqrt{V(Y_{x\ \mathsf{bad}})+V(Y_{x\ \mathsf{good}})}$. The probability that the correct Y_x is lower than an arbitrary wrong Y_x is $\rho=\varphi(-\lambda)$. That is, the expected number of wrong x's with larger Y_x is

$$r=(N_x-1)\varphi(-\lambda) \tag{7.1}$$

So,

$$n=\frac{\lambda^2\sum_\ell a_\ell^2\left[2\left(\frac{q_\ell}{N_x}\right)\left(1-\frac{q_\ell}{N_x}\right)\left(1-\frac{1}{N_\nu}\right)^2+q_\ell\left(p_\ell-\frac{1}{N_\nu}\right)\left(1-\frac{1}{N_\nu}\right)\right]}{\left(\sum_\ell a_\ell q_\ell\left(p_\ell-\frac{1}{N_\nu}\right)\right)^2}$$

By derivating both terms of the fraction with respect to a_ℓ and equaling them, we obtain that the optimal value is reached for

$$a_\ell=a_{\mathsf{opt}}\stackrel{\mathsf{def}}{=}\frac{\left(p_\ell-\frac{1}{N_\nu}\right)}{\left(p_\ell-\frac{1}{N_\nu}\right)+\frac{2}{N_x}\left(1-\frac{1}{N_\nu}\right)\left(1-\frac{q_\ell}{N_x}\right)}$$

Hence, we obtain

$$n=n_{\mathsf{opt}}\stackrel{\mathsf{def}}{=}\frac{\lambda^2\left(1-\frac{1}{N_\nu}\right)}{\sum_\ell a_\ell q_\ell\left(p_\ell-\frac{1}{N_\nu}\right)} \tag{7.2}$$

The attack works as in Algorithm 7.1.

Clearly, the time complexity is nk. The complexity is measured in terms of the number of times the **if** statement is executed. This should have a complexity which is essentially equivalent to executing the phase2 key derivation function. The memory complexity has the order of magnitude of N_x. Another variant is represented in Algorithm 7.2.

Algorithm 7.1 The first attack on WPA (recovering the 7 weak bits of the TK in WPA).

set $I = (2, 3, 13, 14)$ and $I_0 = \{0, 1, 2\}$
initialize the Y_x counters to 0
for $m = 1$ to n **do**
  for $\ell = 1$ to k **do**
    if $g_\ell(h(z^m, \mathsf{IV}^m))$ holds **then**
      compute $\nu = f_\ell(h(z^m, \mathsf{IV}^m))$, the suggested $(\bar{K}[2], \bar{K}[3], \bar{K}[13], \bar{K}[14])$
      compute $x = \pi(\nu)$
      increment Y_x by a_ℓ
    end if
  end for
end for
output $x = \arg\max_x Y_x$

Algorithm 7.2 Another variant of the first attack on WPA.

set $I = (2, 3, 13, 14)$ and $I_0 = \{0, 1, 2\}$
initialize a table y_x^μ to 0
for $\ell = 1$ to k **do**
  for all possible μ such that $g_\ell(\mu)$ holds **do**
    compute $x = \pi(f_\ell(\mu))$
    increment y_x^μ by a_ℓ
  end for
end for
initialize the Y_x counters to 0
for $m = 1$ to n **do**
  for all x **do**
    compute $\mu = h(z^m, \mathsf{IV}^m)$
    increment Y_x by y_x^μ
  end for
end for
output $x = \arg\max_x Y_x$

In the new variant, the time complexity is $N_\mu k + N_x n$ and the memory complexity is $N_\mu N_x$. So, the complexity is

$$c = \min(nk, N_\mu k + N_x n) \tag{7.3}$$

The two complexity curves cross for $n = N_\mu \frac{k}{k - N_x} \approx N_\mu$ when $N_x \ll k$.

For $I = (2, 3, 13, 14)$, we have $N_\nu = 2^{32}$, $N_\mu = 2^{48}$ and $N_x = 2^7$. The complexities with and without using the conditional biases are summarized in Table 7.1. As we can see, when ignoring the conditional biases, we need about 65% more packets, but the complexity is much lower, because k is smaller. So, the conditional biases do not seem to be useful in this case.

7.2.2 The Second Attack on WPA

Let $I_0 = \{0, 1, 2\}$, $I = (15, 2, 3, 14)$ and $x = \mathsf{low1}(\mathsf{TK}[1])$ be the last weak bit. Given the IV and $\nu = (\bar{K}[2], \bar{K}[3], \bar{K}[14], \bar{K}[15])$, we deduce $x = \pi(\nu)$ by

$$\pi(\nu) = \mathsf{high1}((\bar{K}[3] - \bar{K}[2]) \oplus (\bar{K}[15] - \bar{K}[14]))$$

So, we apply the first attack with this I and $N_x = 2$. As $15 \in I$, we have more biases. We have r, n and c from Eq. (7.1), Eq. (7.2) and Eq. (7.3).

For $I = (15, 2, 3, 14)$, we have $N_\nu = 2^{32}$, $N_\mu = 2^{120}$ and $N_x = 2$. The complexities are summarized in Table 7.1. Again, the conditional biases are not very useful. We can also see that this choice of I leads to a better attack than the one from Sec. 7.2.1 in terms of n, but the complexity is slightly higher. This is due to a larger k.

7.2.3 Merging the Attacks

Given two attacks with the sets I^1 (resp. I^2) for recovering independent x^1 (resp. x^2) leading to characteristics Y_{x^1} (resp. Y_{x^2}), c^1 (resp. c^2), n^1 (resp. n^2) and λ^1 (resp. λ^2), one problem is to merge the sorted lists of x^1 and x^2. One can follow the approach by Junod-Vaudenay [JV03]. We sort the pairs following their likelihood ratio, which is obtained by multiplying the likelihood ratio of both terms. We assume that all Y_{x^i} are independent, normally distributed with the variance either $V(Y_{x^i\ \mathsf{bad}})$ or $V(Y_{x^i\ \mathsf{good}}) = V(Y_{x^i\ \mathsf{bad}}) + \Delta V(Y_{x^i})$ and the expected value either $E(Y_{x^i\ \mathsf{bad}})$ or $E(Y_{x^i\ \mathsf{good}}) = E(Y_{x^i\ \mathsf{bad}}) + \Delta E(Y_{x^i})$. Given x^i, the ratio for x^i being the correct value based on the observation Y_{x^i} is

$$\begin{aligned}
\frac{\Pr[Y_{x^i} | x^i\ \mathsf{good}]}{\Pr[Y_{x^i} | x^i\ \mathsf{wrong}]} &= \frac{\frac{1}{\sqrt{2\pi V\left(Y_{x^i\ \mathsf{good}}\right)}} e^{-\frac{\left(Y_{x^i} - E\left(Y_{x^i\ \mathsf{good}}\right)\right)^2}{2V\left(Y_{x^i\ \mathsf{good}}\right)}}}{\frac{1}{\sqrt{2\pi V\left(Y_{x^i\ \mathsf{bad}}\right)}} e^{-\frac{\left(Y_{x^i} - E\left(Y_{x^i\ \mathsf{bad}}\right)\right)^2}{2V\left(Y_{x^i\ \mathsf{bad}}\right)}}} \\
&= \sqrt{\frac{V\left(Y_{x^i\ \mathsf{bad}}\right)}{V\left(Y_{x^i\ \mathsf{good}}\right)}} e^{\frac{\left(Y_{x^i} - E\left(Y_{x^i\ \mathsf{bad}}\right)\right)^2}{2V\left(Y_{x^i\ \mathsf{bad}}\right)} - \frac{\left(Y_{x^i} - E\left(Y_{x^i\ \mathsf{good}}\right)\right)^2}{2V\left(Y_{x^i\ \mathsf{good}}\right)}}
\end{aligned}$$

So, when multiplying some terms of this form for the pairs of values, sorting them by decreasing product is equivalent to sorting them by the decreasing value of

$$\begin{aligned}&\tfrac{1}{2}\left(\tfrac{1}{V_{1b}}-\tfrac{1}{V_{1g}}\right)Y_{x^1}^2+\left(\tfrac{E_{1g}}{V_{1g}}-\tfrac{E_{1b}}{V_{1b}}\right)Y_{x^1}+\tfrac{1}{2}\left(\tfrac{1}{V_{2b}}-\tfrac{1}{V_{2g}}\right)Y_{x^2}^2+\left(\tfrac{E_{2g}}{V_{2g}}-\tfrac{E_{2b}}{V_{2b}}\right)Y_{x^2}\\&=a\left(Y_{x^1}-\beta_1\right)^2+b\left(Y_{x^2}-\beta_2\right)^2\end{aligned}$$

where

$$\begin{array}{lllllll}V_{1g} & = & V(Y_{x^1\ \mathsf{good}}) & & V_{2g} & = & V(Y_{x^2\ \mathsf{good}})\\ V_{1b} & = & V(Y_{x^1\ \mathsf{bad}}) & & V_{2b} & = & V(Y_{x^2\ \mathsf{bad}})\\ \Delta V_1 & = & \Delta V(Y_{x^1}) & & \Delta V_2 & = & \Delta V(Y_{x^2})\\ & & & & & & \\ E_{1g} & = & E(Y_{x^1\ \mathsf{good}}) & & E_{2g} & = & E(Y_{x^2\ \mathsf{good}})\\ E_{1b} & = & E(Y_{x^1\ \mathsf{bad}}) & & E_{2b} & = & E(Y_{x^2\ \mathsf{bad}})\end{array}$$

$$\begin{array}{lllllll}a & = & \tfrac{1}{2}\left(\tfrac{1}{V_{1b}}-\tfrac{1}{V_{1g}}\right) & & b & = & \tfrac{1}{2}\left(\tfrac{1}{V_{2b}}-\tfrac{1}{V_{2g}}\right)\\ \beta_1 & = & \left(\tfrac{V_{1g}E_{1b}-V_{1b}E_{1g}}{\Delta V_1}\right) & & \beta_2 & = & \left(\tfrac{V_{2g}E_{2b}-V_{2b}E_{2g}}{\Delta V_2}\right)\end{array}$$

So we let $Y_{x^1,x^2}=a\left(Y_{x^1}-\beta_1\right)^2+b\left(Y_{x^2}-\beta_2\right)^2$. With the same assumptions as in [JV03], we are back in a situation where Y_{x^1,x^2} is distributed with Generalized-χ^2 distribution [Dav73, Dav80a] (see Appendix A.1 for more details). The average number of wrong (x^1,x^2) pairs with a higher score than the good one is

$$r=\left(N_{x^1}N_{x^2}-1\right)\ \cdot\ \mathsf{Pr}\left(Y_{x^1,x^2\ \mathsf{good}}-Y_{x^1,x^2\ \mathsf{bad}}<0\right)$$

Thus, we define a new random variable

$$\Delta Y_{x^1,x^2}\ \stackrel{\mathsf{def}}{=}\ Y_{x^1,x^2\ \mathsf{good}}-Y_{x^1,x^2\ \mathsf{bad}}=\sum_{m=1}^{2}\sum_{j=b,g}a_{mj}\left[\frac{\left(Y_{x^m j}-\beta_m\right)^2}{V_{mj}}\right]$$

where

$$\begin{array}{lll}a_{1g}=aV_{1g} & a_{1b}=-aV_{1b} & Y_{x^i g}=Y_{x^i\ \mathsf{good}}\\ a_{2g}=bV_{2g} & a_{2b}=-bV_{2b} & Y_{x^i b}=Y_{x^i\ \mathsf{bad}}\end{array}$$

$\Delta Y_{x^1,x^2}$ is a quadratic form in independent normal random variables. It can be expressed as the linear combination

$$\Delta Y_{x^1,x^2}=\sum_{m=1}^{2}\sum_{j=b,g}a_{mj}X_{mj}^2 \tag{7.4}$$

where X_{mj}'s are independent and normally distributed random variables with variance one. We write

$$t_{mj}^2=\frac{\left(E(Y_{mj})-\beta_m\right)^2}{V(Y_{mj})}=t_{mj}'^2\ \cdot\ n$$

The characteristic function of a quadratic form in independent normal random variables $\Delta Y_{x^1,x^2}$

is given by [Dav73], where

$$\varphi_{\Delta Y_{x^1,x^2}}(u) = E(e^{iu\Delta Y_{x^1,x^2}}) = \frac{e^{iu\left(\sum_{m=1}^{2}\sum_{j=b,g}\frac{a_{mj}t_{mj}^2}{1-2iua_{mj}}\right)}}{\prod_{m=1}^{2}\prod_{j=b,g}(1-2iua_{mj})^{\frac{1}{2}}}$$

If $E(|\Delta Y_{x^1,x^2}|)$ is finite, it follows from [GP51] that

$$F_{\Delta Y_{x^1,x^2}}(w) = \mathsf{Pr}(\Delta Y_{x^1,x^2} < w) = \frac{1}{2} - \int_{-\infty}^{\infty} \mathsf{Im}\left(\frac{\varphi_{\Delta Y_{x^1,x^2}}(u)e^{-iuw}}{2\pi u}\right) du$$

Substituting what we have, one derives

$$F_{\Delta Y_{x^1,x^2}}(0) = \mathsf{Pr}(\Delta Y_{x^1,x^2} < 0) = \frac{1}{2} - \int_{-\infty}^{\infty} \mathsf{Im}\left(\frac{e^{iu\left(\sum_{m=1}^{2}\sum_{j=b,g}\frac{a_{mj}t_{mj}^2}{1-2iua_{mj}}\right)}}{2\pi u\prod_{m=1}^{2}\prod_{j=b,g}(1-2iua_{mj})^{\frac{1}{2}}}\right) du$$

Finally, setting r, the value of n can be numerically computed.

It might be of interest to evaluate n analytically. In Eq. (7.4), the X_i^2's follow the non-centralized χ^2 distribution. Our experiment revealed that their non-centrality parameters are large. Let n_i and t_i^2 be their corresponding degrees of freedom and non-centrality parameters respectively. It was shown in [Mui05] (see pages $22-24$ and problem 1.18) that when $n_i \to \infty$ or $t_i^2 \to \infty$, the non-centralized χ^2 random variable can be approximated by the normal distribution with the same expected value and variance. Using this approach, the above integral can be avoided. Hence,

$$\begin{aligned} E(\Delta Y_{x^1,x^2}) &\approx \sum_{m=1}^{2}\sum_{j=b,g} a_{mj}\left(1+t_{mj}^2\right) \\ V(\Delta Y_{x^1,x^2}) &\approx \sum_{m=1}^{2}\sum_{j=b,g} 2a_{mj}^2\left(1+2t_{mj}^2\right) \end{aligned}$$

To find n, we need to solve the following equation.

$$\left(\frac{-E(\Delta Y_{x^1,x^2})}{\sqrt{V(\Delta Y_{x^1,x^2})}}\right) = \varphi^{-1}\left(\frac{r}{N_{x^1}N_{x^2}-1}\right)$$

Thus, we derive

$$n \approx \left[\frac{1}{\mu}\varphi^{-1}\left(\frac{r}{N_{x^1}N_{x^2}-1}\right)\right]^2$$

where

$$\mu = \frac{\sum_{m=1}^{2}\sum_{j=b,g} a_{mj}t'_{mj}}{\sqrt{\sum_{m=1}^{2}\sum_{j=b,g} 4a_{mj}^2 t'_{mj}}}$$

Table 7.1: The complexities of several attacks to recover $\log_2 N_x$ bits from the TK. We compare them when including the conditional biases and without. We provide the number of packets n, the running time complexity c, the expected number r of the better wrong values, as well as the parameters k, λ and N_ν. Except when $N_x = 2$ for which it would not make any sense, we target $r = \frac{1}{2}$ (that is, the correct value has the higher score in half of the cases). We used $I_0 = \{0, 1, 2\}$.

	ref.	I	n	c	r	N_x	k	λ	N_ν	N_μ	cond.
1u		$(2, 3, 13, 14)$	$2^{42.10}$	$2^{42.10}$	$\frac{1}{2}$	2^7	1	2.66	2^{32}	N^6	without
1c		$(2, 3, 13, 14)$	$2^{41.38}$	$2^{53.10}$	$\frac{1}{2}$	2^7	$2^{11.72}$	2.66	2^{32}	N^8	with
2u		$(15, 2, 3, 14)$	$2^{40.38}$	$2^{45.38}$	$\frac{1}{4}$	2	2^5	0.67	2^{32}	N^{15}	without
2c		$(15, 2, 3, 14)$	$2^{39.12}$	$2^{55.85}$	$\frac{1}{4}$	2	$2^{16.73}$	0.67	2^{32}	N^{17}	with
3u	merge 1u+2u		$2^{41.83}$	$2^{46.87}$	$\frac{1}{2}$	2^8					without
3c	merge 1c+2c		$2^{41.22}$	$2^{57.99}$	$\frac{1}{2}$	2^8					with
4u		$(15, 2, 3, 13, 14)$	$2^{51.72}$	$2^{57.72}$	$\frac{1}{2}$	2^8	2^6	2.88	2^{40}	N^{17}	without
4c		$(15, 2, 3, 13, 14)$	$2^{51.05}$	$2^{72.69}$	$\frac{1}{2}$	2^8	$2^{21.64}$	2.88	2^{40}	N^{19}	with

We can use these merging rules to merge the two previous attacks, i.e., setting $c = c^1 + c^2$ by using Eq. (7.3) for c^1 and c^2. We obtain the results from Table 7.1.

Table 7.1 shows the complexities when merging the previous attacks to recover the 8 weak bits of the TK. We compare them with the attack using a merged set I directly. As we can see, merging the attacks with small I's (reference 3) is much better than making a new attack with a merged I (reference 4).

7.2.4 A Temporary Key Recovery Attack on WPA

The results from [MRH04] lead to an "easy" attack on WPA: guess the 96-bit PPK and the 8 weak bits of the TK with an average complexity of 2^{103} until it generates the correct keystream. Then, guess the 96-bit PPK of another packet in the same segment (with the weak bits already known). Then, apply the method of [MRH04] to recover the TK. We improve this attack by recovering the weak bits of the TK separately: we know from Table 7.1 that we can recover the weak bits of TK by using 2^{42} packets. After having recovered the weak bits, we note that the 96-bit PPK is now enough to recalculate the RC4KEY. So, we can do an exhaustive search on the PPK for a given packet until we find the correct one generating the packet. This works with an average complexity of 2^{95}. We do it twice to recover the PPK of two packets in the same segment. Given these two PPK sharing the same IV32, we recover the TK by using the method of [MRH04]. Therefore, we can recover the temporary key TK and decrypt all the packets with complexity 2^{96}. The number of packets needed to recover the weak bits is 2^{42}.

7.3 Distinguishing WPA

As first mentioned by Mantin during the [SVV11] presentation in Estonia, RC4 can be distinguished using N packets [MS01] and since WPA's output is already an output of RC4, it can be simply distinguished from random using a few packets. However, the distinguisher of [MS01], based on the bias of z_2, can not distinguish two protocols that are both using RC4. In this section, we are using the previous biases on RC4 together from some weaknesses on the structure of WPA and mount a distinguishing attack on WPA. This distinguisher is also capable of distinguishing WPA from other protocols using RC4.

The first attack can be turned into a distinguisher as follows: The expected value and the variance of the correct Y_x are

$$E(Y_{x\ \mathsf{good}}) = E(Y_{x\ \mathsf{bad}}) + \lambda\sqrt{V(Y_{x\ \mathsf{bad}}) + V(Y_{x\ \mathsf{good}})}$$
$$V(Y_{x\ \mathsf{good}}) = V(Y_{x\ \mathsf{bad}}) + \Delta V(Y)$$

Let's extend our notations by defining

$$\gamma = \left(\frac{V(Y_{x\ \mathsf{good}})}{V(Y_{x\ \mathsf{bad}})}\right)$$

The random variable Y_x of the good counter is larger than

$$T = E(Y_{x\ \mathsf{bad}}) + \lambda'\sqrt{V(Y_{x\ \mathsf{bad}}) + V(Y_{x\ \mathsf{good}})}$$

with probability $\varphi\left((\lambda-\lambda')\sqrt{1+\frac{1}{\gamma}}\right)$. Now, if we replace the WPA packets by some random sequences, all the counters have expected value $E(Y_{x\ \mathsf{bad}})$ and variance approximately $V(Y_{x\ \mathsf{bad}})$. The probability that a given counter exceeds T is $\varphi\left(-\lambda'\sqrt{1+\gamma}\right)$. The probability that any counter exceeds this is lower than $N_x\varphi\left(-\lambda'\sqrt{1+\gamma}\right)$. So, the condition $\max_x Y_x > T$ makes a distinguisher of the same n and c as in the first attack and with $\mathsf{Adv} \geq \beta$, where

$$\beta = \varphi\left((\lambda-\lambda')\sqrt{1+\frac{1}{\gamma}}\right) - N_x\varphi\left(-\lambda'\sqrt{1+\gamma}\right) \tag{7.5}$$

Finally, we find the optimal λ' which maximizes the advantage.

$$\lambda' = \frac{\sqrt{\left(1+\frac{1}{\gamma}\right)^2\lambda^2 + \left(\gamma-\frac{1}{\gamma}\right)\left[\left(1+\frac{1}{\gamma}\right)\lambda^2 + 2\ln\left(N_x\sqrt{\gamma}\right)\right]} - \left(1+\frac{1}{\gamma}\right)\lambda}{\left(\gamma-\frac{1}{\gamma}\right)}$$

We use the same values as before and target $\mathsf{Adv} \geq \frac{1}{2}$. We use Eq. (7.2) for n, Eq. (7.3) for c and Eq. (7.5) for a lower bound β of the advantage. The performances of the distinguishers are summarized on Table 7.2. Again, the attack based on $I = (15, 2, 3, 14)$ is better in terms of the number of packets, but not in terms of the complexity. It works using $2^{41.23}$ packets and complexity $2^{46.23}$. The one based on $I = (2, 3, 13, 14)$ works with 50% more packets ($2^{41.83}$) with no conditional biases, but with a much better complexity of $2^{41.83}$.

Table 7.2: The complexities of several distinguishers for WPA. We compare them when including the conditional biases and without. We provide the number of packets n, the running time complexity c, the bound on the advantage β, as well as parameters k, λ and N_ν. We target $\beta = \frac{1}{2}$. We used $I_0 = \{0, 1, 2\}$.

	I	n	c	β	N_x	k	λ	N_ν	N_μ	cond. biases
1u	$I = (2, 3, 13, 14)$	$2^{41.83}$	$2^{41.83}$	0.5	2^7	1	2.42	2^{32}	N^6	without
1c	$I = (2, 3, 13, 14)$	$2^{41.11}$	$2^{52.83}$	0.5	2^7	$2^{11.72}$	2.42	2^{32}	N^8	with
2u	$I = (15, 2, 3, 14)$	$2^{41.23}$	$2^{46.23}$	0.5	2	2^5	1.28	2^{32}	N^{15}	without
2c	$I = (15, 2, 3, 14)$	$2^{40.97}$	$2^{57.70}$	0.5	2	$2^{16.73}$	1.28	2^{32}	N^{17}	with

Chapter 8

Cryptanalysis of the WEP Protocol

For many people, breaking WEP is similar to beating a dead horse. Unfortunately, it is still widely used in many countries. Hence, we believe further analysis into this protocol is still of great interest. Moreover, it also provides a better understanding of the security of RC4. In most previous analysis of WEP, there is no *concrete* theory behind most of the claims, i.e., there is no way to test whether the attacks work, except if we implement them again and check the results. If a new correlation is found for RC4 that is exploitable whether against WEP or WPA, someone should implement the attacks and go through a very time consuming experimental analysis to check the new correlation. Also, many biases already existing in the literature are overestimated or the probabilities are not properly computed. Many parameters were set incorrectly or heuristically without any proper justification of their origin.

We performed more than one year of extensive theoretical and experimental analysis of all the biases we used for RC4 (see Chapters 5, 6) and the complexities we derived in Chapter 7. All the attacks were fully implemented and match the theoretical analysis.

We first elaborate the FMS attack on WEP and then explain how we can achieve better results by using the Klein-Improved, the Korek and the SVV_10 correlations. To avoid all the heuristic settings in the literature against WEP, we compute all the parameters of our attacks theoretically and derive an optimized attack on the WEP protocol. We call this attack *Tornado attack* due to the similarity between the distribution of our counters and the distribution of Tornadoes in real life. Finally, we deploy a sequential distinguishing approach to reduce the data complexity of our attack.

In the initial attacks on WEP such as the FMS attack and the the Klein attack, the attacker recovers each byte of the key individually. Later, it was found in [VV07, TWP07] that we can use the repetition of the key to increase the success probability of the attacks. In fact, according to the $\bar{K}[i+16j] = \bar{K}[i] + j\bar{K}[15]$ relation, if $\bar{K}[15]$ is known, the biases for $\bar{K}[i]$ can be merged with $\bar{K}[i+16j]$ to obtain a higher success probability. Therefore, both papers recover the sum of the key bytes instead. This was not the case for the FMS and the Klein attacks.

8.1 The Fluhrer, Mantin and Shamir (FMS) Attack

In 1995, Roos [Roo95] and Wagner [Wag95] found a class of weak keys for RC4. This class was derived from some linear properties of the PRGA and Roos correlation for the KSA (see Sec. 4.1.1). Depending on the value of the first 3 bytes of the key, the fourth byte might be recovered with their attack. They could not exploit this weakness against any protocol using RC4, because at that time WEP protocol did not exist. In fact, it was proposed 4 years later in 1999 [IEE99a].

Later, Fluhrer, Mantin and Shamir [FMS01] used Roos and Wagner class of weak keys and mounted a devastating attack on the WEP protocol. They claimed that 4 million packets would be enough to recover the WEP secret key with probability 50% with incremental IV's. Later in [SIR02, SIR04], the authors showed that in fact between 4 million to 6 million packets are required to break a WEP key with the FMS attack. Consequently, many vendors restricted the class of weak keys used by the FMS attack to avoid RC4 weaknesses in the WEP mode.

In the following, we are going to describe this attack. The conditions, the assumptions, the key recovery relation and the success probability are listed in Algorithm 8.1. We name this algorithm FMS-Improved, as in the initial FMS attack in [FMS01], it was assumed that $t = i - 1$ and the key bytes are recovered instead of their sum.

Algorithm 8.1 The FMS-Improved Attack

Success Probability: $P_{\mathsf{FMSI}}(t)$
Assumptions: (see Figure 8.1)
1: $S_t[0] = \cdots = S_{N-1}[0] = i$
2: $S_t[1] = \cdots = S_{N-1}[1] = 0$
3: $S_t[j_i] = \cdots = S_{i-1}[j_i] = S_i[i] = \cdots = S_{N-1}[i]$
Conditions: $z_1 \notin \{0, i\}$ and $(S_t^{-1}[z_1] < t + 1$ or $S_t^{-1}[z_1] > i - 1)$
Key recovery relation: $\bar{K}[i] = S_t^{-1}[z_1] - \sigma_i(t)$

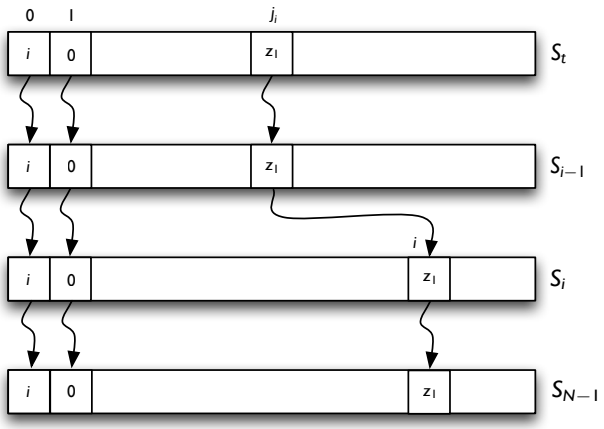


Figure 8.1: The RC4 state update in the FMS-Improved attack.

If we focus on the PRGA, at the first round $S_{N-1}[1]$ and $S_{N-1}[0]$ are swapped. Then, the value

of z_1 is computed as $z_1 = S'_1[S'_1[1] + S'_1[0]] = S'_1[i] = z_1$. According to step 6 of the KSA, we have $j_i = S_{i-1}[i] + j_{i-1} + K[i]$. The index of z_1 should be less than $t+1$ or more than $i-1$, otherwise z_1 would be swapped in S_t to S_{i-1} state updates. Hence, the FMS attack recovers the key bytes as

$$K[i] = S_t^{-1}[z_1] - S_{i-1}[i] + j_{i-1}$$

Indeed, we start recovering $K[3]$ first and then we update the state to S_3. Consequently, we recover $K[4], \ldots, K[15]$ similarly. This relation can be improved by recovering the sum of the key bytes (FMS-Improved attack) instead. Using Lemma 3.3, the key recovery relation becomes $\bar{K}[i] = S_t^{-1}[z_1] - \sigma_i$; To recover $\bar{K}[i]$, it is not necessary any more to recover $\bar{K}[i-1]$ first. The success probability of the attack using Lemma 3.4 is

$$P_{\mathsf{FMSI}} = R_2^3(i,t) \ \cdot \ P_B(i,t) + \frac{1}{N-1}\left(1 - R_2^3(i,t)\right)\left(1 - P_B(i,t)\right)$$

where

$$R_2^3(i,t) = r_2(i) \ \cdot \ P_A^3(i,t) + \tfrac{1}{N}\left(1 - r_2(i) \ \cdot \ P_A^3(i,t)\right)$$
$$r_2(i) = \left(\tfrac{N-3}{N}\right)^{N-i-1}$$

8.2 Tornado Attack on WEP

We use a similar strategy as the attack on WPA. As the values of $\bar{K}[0], \bar{K}[1]$ and $\bar{K}[2]$ are known, we update the state to S_2. We first recover the value of $\bar{K}[15]$ by using all the biases we have. In fact, we can merge the biases for $\bar{K}[15], \ldots, \bar{K}[18]$ and $\bar{K}[31], \ldots, \bar{K}[34]$, as the first 3 bytes of the key are known. As the biases $\bar{K}[31], \ldots, \bar{K}[34]$ vote for $2\bar{K}[15]$, we vote for two distinct values in $\mathbf{Z}_{256}$ in the table of $\bar{K}[15]$. Then, we recover $\bar{K}[3]$ by merging all the biases for $\bar{K}[3]$ and $\bar{K}[19]$. In practice, we do not use any bias for $i > 34$, since after $i = 34$, the probabilities are very close to the uniform distribution and also we can not predict the keystream. Then, we update the state to S_3 and vote for $\bar{K}[4]$ and $\bar{K}[20]$ and we merge them. We repeat this procedure until $\bar{K}[14]$. Finally, we check the recovered key. If it is not the correct key, then we pick the second best element in the sorted list of $\bar{K}[3]$ and repeat the same procedure. So, at the end our approach is recursive.

To be more precise, we apply the first attack on WPA with $x = \nu$: we only recover key bytes which are the same for all packets. This attack produces a ranking of possible x's in the form of a list $\mathcal{L}$ by decreasing order of likelihood (see Algorithm 8.2).

Let $\Pi_i = \Pi(i|0, \ldots, i-1, 15)$ for $i = 3, \ldots, i_{\max}$ and $\Pi_{15} = \Pi(15|0,1,2)$ be the table of biases used by the attack on $\bar{K}[i]$. Similarly, let $N_x = N_\nu = N$ and r_i, c_i be their parameters following Eq. (7.1,7.3). Let R_i be the rank of the correct $\bar{k}_i$ value in $\mathcal{L}_i$. Let's define a random variable $U_{ij} = 1_{(Y_{x^i \ \mathrm{good}} < Y_{x^i \ \mathrm{bad}^j})}$, where $Y_{x^i \ \mathrm{bad}^j}$ is the j-th bad counter in attacking $\bar{K}[i]$. Hence, we have

$$R_i = \sum_{j=1}^{N_x - 1} U_{ij}$$

Algorithm 8.2 An optimized attack against the WEP protocol

compute the ranking $\mathcal{L}_{15}$ for $I = (15)$ and $I_0 = \{0, 1, 2\}$
truncate $\mathcal{L}_{15}$ to its first ρ_{15} terms
for each $\bar{k}_{15}$ in $\mathcal{L}_{15}$ **do**
    run recursive attack on input $\bar{k}_{15}$
end for
stop: attack failed
recursive attack with input $(\bar{k}_{15}, \bar{k}_3, \ldots, \bar{k}_{i-1})$**:**
If input is only $\bar{k}_{15}$, set $i = 3$.
if $i \leq i_{\max}$ **then**
    compute the ranking $\mathcal{L}_i$ for $I = (i)$ and $I_0 = \{0, \ldots, i-1, 15\}$
    truncate $\mathcal{L}_i$ to its first ρ_i terms
    for each $\bar{k}_i$ in $\mathcal{L}_i$ **do**
        run recursive attack on input $(\bar{k}_{15}, \bar{k}_3, \ldots, \bar{k}_{i-1}, \bar{k}_i)$
    end for
else
    for each $\bar{k}_{i_{\max}+1}, \ldots, \bar{k}_{14}$ **do**
        test key $(\bar{k}_3, \ldots, \bar{k}_{14}, \bar{k}_{15})$ and stop if correct
    end for
end if

The expected value and the variance of this random variable can be computed as follows:

$$r_i = E(R_i) = (N_x - 1)\varphi(-\lambda_i)$$

and (8.1)

$$E(R_i^2) = E(R_i) + (N_x - 1)(N_x - 2) \cdot E(U_{i1}.U_{i2})$$

where

$$E(U_{i1}.U_{i2}) = \frac{1}{\sqrt{2\pi V(Y_{x^i\ \mathsf{good}})}} \int_{-\infty}^{\infty} e^{-\frac{\left(Y - E(Y_{x^i\ \mathsf{good}})\right)^2}{2V(Y_{x^i\ \mathsf{good}})}} \left(1 - \varphi\left(\frac{Y - E(Y_{x^i\ \mathsf{bad}})}{\sqrt{V(Y_{x^i\ \mathsf{bad}})}}\right)^2\right) dY$$

This finally yields

$$V(R_i) = (N_x - 1)\varphi(-\lambda_i) + (N_x - 1)(N_x - 2)\ .\ E(U_{i1}.U_{i2}) - (N_x - 1)^2\varphi(-\lambda_i)^2 \tag{8.2}$$

In [SVV11], U_{i1} and U_{i2} were incorrectly assumed to be independent, leading to

$$V(R_i) \approx (N_x - 1)\varphi(-\lambda_i)(1 - \varphi(\lambda_i)) \approx r_i$$

which did not match our experiment. Now, the fundamental question is what would be the distribution of R_i. This is discussed in the next section.

8.2.1 Analysis Based on Pólya Distribution

In [SVV11], it was assumed that the distribution of R_i is normal. Running a few experiments, we noticed that in fact it is following a distribution very close to the Poisson distribution. An

amazing observation was that the variance of the distribution was much higher than the expected value. A number of distributions have been devised for series in which the variance is significantly larger than the mean [Ans50, Fel43, Ney39], frequently on the basis of more or less complex biological models [BF53]. The first of these was the negative binomial, which arose in deriving the Poisson series from the point binomial [Stu07, Whi14]. We use a generalized version of negative binomial distribution called the Pólya distribution (see Appendix A.2 for more details). The main application of the Pólya distribution is in *Tornado Outbreaks* [Tho63] and *Hail Frequency* analysis [Tho57].

In most climates, the probability of hail is small. If the hail frequency ranges on an interval $f_1 < f < f_2$ for all climates, it is observed that for values of f near f_1 the hail storms are quite scattered through each year. For this case, the hail storms might be considered independent of each other. In this instance, the series of annual frequencies of the hail events might be expected to follow the Poisson distribution of rare events. On the other hand, if the mean hail frequency is near f_2, then it seems reasonable to assume that the successive hail storms may no longer be independent and so, if one storm had hail, the next storm would be more likely to have hail as well. The introduction of dependence between successive storms leads in a special fashion to the negative binomial distribution [Tho57]. Similarly, tornadoes tend to cluster within years and follow a Pólya process rather than a Poisson process in areas where the frequency of occurrence is high [Tho63].

This observation led us to find out that R_i is in fact following the Pólya distribution (see Appendix A for the definition of the Pólya distribution and some of its properties). To be more precise, if two events occur with Poisson distribution and their expected value is very low, then it can be assumed that those events are happening independently. On the other hand, for the Poisson events with high expected values (approximated as normal), the occurrence of the former event may increase the probability of the latter. In such cases, the overall distribution would be the Pólya. Regarding the current problem, the events $(Y_{x^i \text{ good}} < Y_{x^i \text{ bad}^j})$ and $(Y_{x^i \text{ good}} < Y_{x^i \text{ bad}^{j'}})$ are not independent. Therefore, they tend to follow the Pólya distribution. As $E(R_i)$ and $V(R_i)$ are known from Eq. (8.1, 8.2), the values p_i and r_i for attacking $\bar{K}[i]$ can be simply computed by

$$p_i = \left(1 - \frac{E(R_i)}{V(R_i)}\right) \qquad \text{and} \qquad r_i = \left(\frac{E(R_i)^2}{V(R_i) - E(R_i)}\right)$$

As a proof of concept, we have sketched the probability distribution of R_3 for 5 000 packets. The corresponding parameters for the Pólya distribution would be $p = 0.9839$ and $r = 0.356$ (see Figure 8.2). As can be observed, those two distributions are extremely close. Also,

$$u_i \stackrel{\text{def}}{=} \mathsf{Pr}[R_i \leq \rho_i - 1] = 1 - I_{p_i}(\rho_i, r_i)$$

where I is the regularized incomplete beta function. Overall, the success probability is

$$u = u_{15} \prod_{i=3}^{i_{\max}} u_i$$

and the complexity is

$$c = c_{15} + \rho_{15}\left(c_3 + \rho_3\left(c_4 + \rho_4\left(\cdots c_{i_{\max}} + \rho_{i_{\max}} N^{14-i_{\max}} \cdots\right)\right)\right)$$

To be able to compare our results with the state of the art, we set $u = 50\%$. To approximate the optimal choice of ρ's, let $i_{\max} = 14$. We have to deal with the following optimization problem:

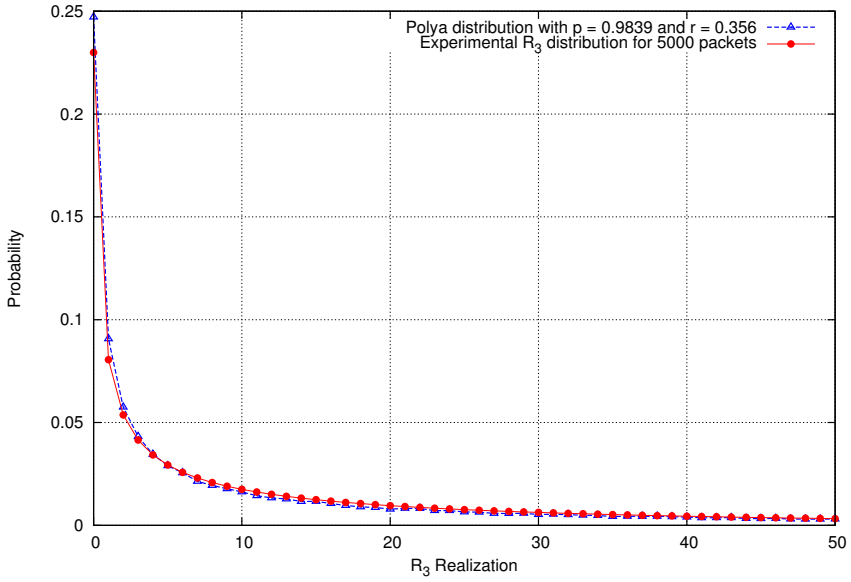


Figure 8.2: R_3 distribution using 5 000 packets following the Pólya distribution

Minimize c in terms of ρ_i's, with the constraint that $u = \prod_{i=3}^{15} (1 - I_{p_i}(\rho_i, r_i)) = \frac{1}{2}$

To solve this optimization problem, we deploy three distinct approaches:

– To obtain the probability 50%, we let the probabilities u_i's to be equal for all $i \in \{3, \ldots, 15\}$. Hence, we set

$$(1 - I_{p_i}(\rho_i, r_i)) = 2^{\left(\frac{-1}{i_{\max}-1}\right)} = 0.9481$$

and we find the corresponding ρ_i's. This approach does not yield the optimal solution, but at least it gives a benchmark on what we should expect.

– Another approach is to use Lagrange multipliers to find the optimal solution. We used the fmincon function in Matlab with the Sequential Quadratic Programming [NW06] (SQP) algorithm as the default algorithm to compute the local minimum. This algorithm was very fast and stable compared to the Genetic algorithm being explained next. As this algorithm needs a starting point x0 for its computations, we used the GlobalSearch class which iterates the fmincon function multiple times using random vectors for x0. Simultaneously, it checks how the results merge towards the global minimum. The drawback of any Lagrange multiplier approach is that the algorithm should be fed with a continuous objective function. This is because it has to compute some derivatives. As we need integer values for ρ_i's in practice, we had to relax the outputs by the ceil function to round up the ρ_i's found by this approach. Therefore, it does not guarantee that the optimal solution is found at the end, but it finds a complexity very close to the optimal. As our experiment revealed, this algorithm most often sets $\rho_{14} = N$. So, using this approach, $i_{\max} = 13$ and we do not often need to vote for $\bar{K}$[14].

– The last approach is to find an algorithm which can handle discrete functions, i.e., it accepts integers as input. One option is to use the Genetic algorithms. We used the ga function in Matlab for this purpose. As these algorithms are evolutionary, the drawback is that with the same parameters, each run outputs different results. So, we had to run the algorithm multiple times and pick the best solution. The other drawback is that it will find a local minimum and does not guarantee to find the global optima. As can be observed in Figure 8.3, this method is not as stable as the other approaches. Moreover, the experiment time is much longer compared to the other methods. To obtain a stable result, the parameters of the Genetic algorithm should be set carefully enough. This approach often yields a high value for ρ_{15}, but it is often less than N.

Moreover, using the empirical distribution of R_i's and by deploying the Genetic algorithm approach, we computed the experimental curve of the complexity. We have depicted the result of all these three approaches in Figure 8.3.

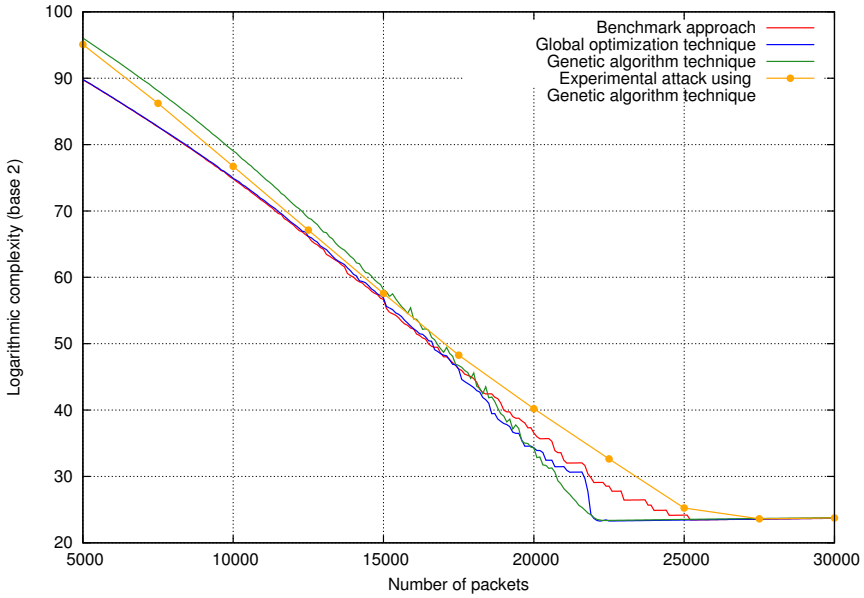


Figure 8.3: The theoretical and experimental logarithmic complexities in terms of the data complexity of breaking a WEP key with probability at least 50% with respect to three distinct optimization approaches: the Benchmark approach, the Global optimization technique and the Genetic algorithm technique.

We call the optimized key ranking attack on RC4, *"Tornado Attack"*, as R_i's follow exactly the same distribution as tornadoes.

Recovering $\bar{K}[15]$ is a crucial step in WPA and WEP attacks. We compare the theoretical and experimental success probability of recovering $\bar{K}[15]$ as the first element in the sorted list. In [SVV11], it is assumed that $Y_{x\ \mathsf{good}} - Y_i$ is independent for all bad i's and was deduced that the good x had a top Y_x with probability $(1 - \varphi(-\lambda))^{N-1}$. Running some experiments, we observed incoherent results which invalidate this model. Figure. 8.4 represents this success probability with respect to the number of packets, theoretically and experimentally. As we already know that the

distribution of the rank is the Pólya distribution, we obtain

$$\Pr[R_{15} = 0] = (1 - p_{15})^{r_{15}}$$

The difference between these two curves are coming from the dependency between biases. In all our analysis, we assumed that the biases are independent, which is not a precise assumption. This difference can be observed in Figure 8.4.

For the attacks against WEP and WPA, we used the biases up to $\bar{K}[34]$. for any $i > 34$, the probabilities are getting very close to the uniform distribution. It can still improve the overall success rate of the attack, but this improvement is not significant and it further increases the computational cost of the attack. The IV's are picked pseudo-randomly using the SNOW 2.0 stream cipher. We used the Klein-Improved attack, the Korek biases and the SVV_10 bias for the attacks on WEP and WPA.

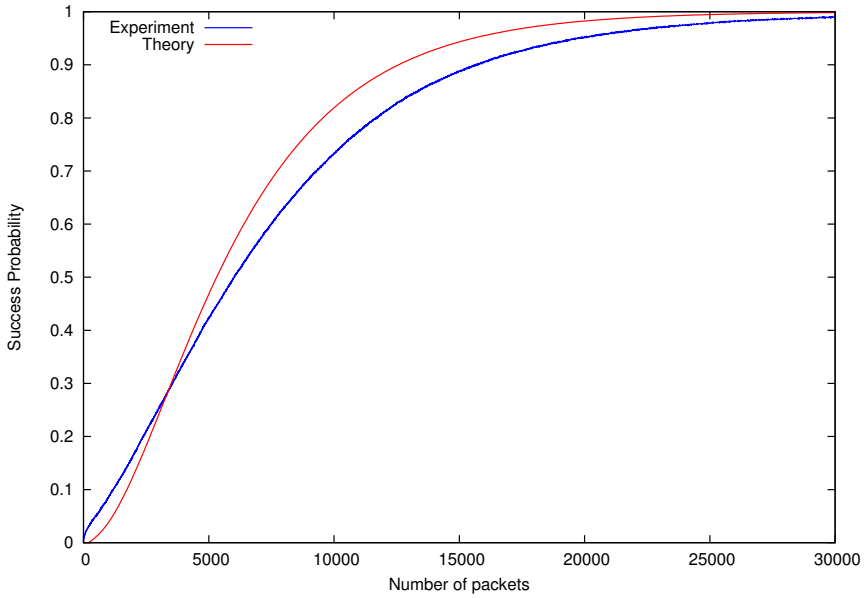


Figure 8.4: The success probability of recovering $\bar{K}[15]$ as the top element in the voted list in theory and practice.

8.2.2 Comparison with Aircrack-ng

Aircrack-ng [DO11] is a WEP and WPA-PSK keys cracking program that can recover keys once enough data packets have been captured. It implements the standard FMS attack [FMS01] along with some optimizations like the Korek attacks [Kor04a, Kor04b], as well as the PTW attack [TWP07]. In fact, it currently has the implementation of state of the art attacks on WEP and WPA. In this section, we compare our results with Aircrack-ng 1.1.

Aircrack-ng does not deploy the recursive algorithm we use. Hence, it is not straightforward to compare our results. A fair approach for comparison is to compare the success probability of both when only the first key in the sorted list is checked i.e., $\rho_i = 1$ for all i. If that fails, the entire key recovery attack will fail. This way, we do not enter the recursive loop. We sketched the

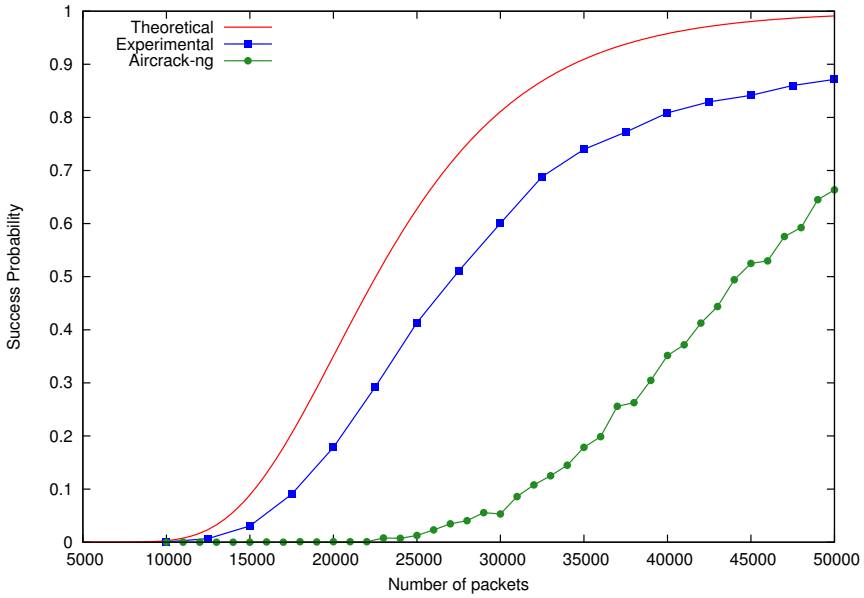


Figure 8.5: The success probability of recovering a WEP key versus the number of packets, when only the first key in the voted list is checked i.e., $\rho_i = 1$ for all i.

corresponding curve for our attack and Aircrack-ng in Figure 8.5. To obtain the success probability of 50%, our attack requires 27 500 packets in average, while Aircrack-ng requires 44 000 packets. With 27 500 packets, Aircrack-ng has only 3% success rate to recover the correct WEP key. With 44 000 packets, our attack can recover a WEP key with probability close to 1. We also sketch the curve of the theoretical success probability in the same figure. The distance between the theoretical result and the experimental one is coming from the dependency between the biases in Table 5.1 and 6.1. This dependency was already observed in Figure 8.4, when recovering $\bar{K}[15]$. Hence, if the same biases are used to recover $\bar{K}[i]$'s for $i \in \{3, \ldots, 15\}$, the distance would accumulate and it makes sense to have such a distance between the theoretical and the experimental curves. One approach to reduce the distance is to discard the biases that do not contribute significantly to the success probability. This can be achieved by discarding a few of the Korek attacks with very low densities.

We also draw the success probability versus the number of packets without the Korek and the SVV_10 biases in Figure 8.6. Note that without these biases, the same analysis with 22 500 packets (50% success rate in theory including all the biases) yields only 18% success rate. So, these biases make a huge difference in this case.

8.2.3 The Sequential Distinguishing Approach

Up to this point, it was assumed that a *fixed* number of packets is given to the adversary and the aim was to maximize the success probability. Changing the perspective, we can look at the problem as fixing the success probability and searching for the minimum average number of packets to gain that probability. This idea was used initially by Davies and Murphy [MD95] to decrease the complexity of their attack against DES. With this type of model in mind, the notion of $n_{\max}$-*limited generic sequential non-adaptive distinguishers* was defined by Junod in [Jun03], where $n_{\max}$ is an

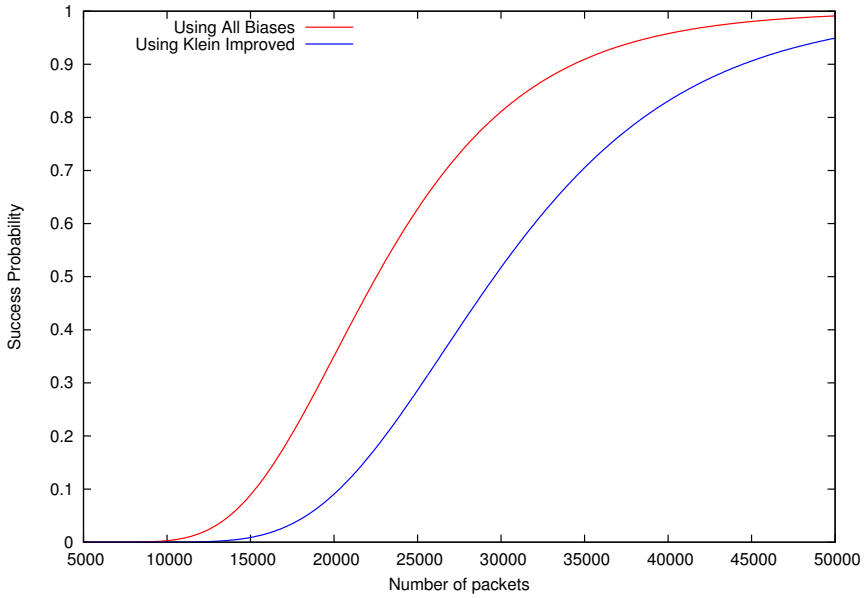


Figure 8.6: The success probability of recovering a WEP key versus the number of packets with the Klein-Improved bias only, when only the first key in the voted list is checked. It also depicts the number of packets by using all the biases.

upper bound for the allowed number of packets in our context. Indeed, we can use a sequential distinguisher for key recovery.

Mapping the definition of an $n_{\max}$-limited generic sequential non-adaptive distinguisher in [Jun03] to our attack, the new attack works as follows: the attacker waits for some small number of packets to be eavesdropped from the channel and then runs the attack from the previous section. If it fails, then it waits for more packets to come and runs the attack again. This procedure is iterated again and again. The attacker would stop when she finds the correct key or the threshold $n_{\max}$ number of packets is reached. If the former occurs, it outputs 1 (success), otherwise it outputs 0 (failure). This attack mode was already used in Aircrack-ng. We refer to it as the *"interactive mode"* from now on.

This approach turns out to be more efficient in terms of the number of packets compared to other types of distinguishers. In fact, Siegmund [Sie85] has proved the following theorem (see [Jun03] for details).

Theorem 8.1. *For a simple hypothesis testing against a simple alternative with independent and identically distributed observations, a sequential probability ratio test is optimal in the sense of minimizing the expected number of samples among all tests having no larger error probabilities.*

Using this technique, we can decrease the average number of packets to reach the success probability of 50%. In fact, we could drop the data complexity of our fastest attack (i.e., with all $\rho_i = 1$) in Figure 8.5 from 27 500 to 22 500 packets in average using this approach to gain the success probability of 50%. This attack runs in less than 3 seconds on an ordinary PC. We give another example to illustrate how the number of packets can be dropped using this technique.

Using 23 000 packets and the attack from the previous section, we computed the almost optimized

ρ_i's derived from the Genetic algorithm approach in practice to gain the success probability of 50%. We set

$$\begin{array}{lllll} \rho_3 = 2 & \rho_4 = 1 & \rho_5 = 1 & \rho_6 = 2 & \rho_7 = 2 \\ \rho_8 = 1 & \rho_9 = 2 & \rho_{10} = 1 & \rho_{11} = 1 & \rho_{12} = 4 \\ \rho_{13} = 2 & \rho_{14} = 86 & \rho_{15} = 1 & & \end{array}$$

Next, we run the attack in interactive mode with the above ρ_i's for a lot of WEP keys and find the minimal value of $n_{\max}$ which yields 50% success rate. Our experiments showed that $n_{\max} = 22\,000$. Consequently, we run the same attack in interactive mode with $n_{\max} = 22\,000$ for recovering different WEP keys K_i leading to some n_i to succeed. Then, we compute the statistical average of the number of packets n_i when it succeeds and $n_{\max}$ for the attacks which fail. The average number of packets we obtained in practice was 19 800 packets. This attack runs in less than a minute on an ordinary PC. If we have less number of packets at our disposal, the attack will run for a larger period.

Chapter 9

Conclusion

We revisited the statistical attacks on RC4, with applications to WEP and WPA protocols. We provided a precise proof for almost all useful correlations existing for RC4 which are exploitable against WEP and WPA.

We deployed a framework to handle pools of biases for RC4 which can be used to break WEP and WPA protocols. In the case of the 8 weak bits of the TK of WPA, we have shown a simple distinguisher and a partial key recovery attack working with 2^{42} packets and a practical complexity. This can be used to improve the attack by Moen-Raddum-Hole [MRH04] to mount a full temporary key recovery attack of complexity 2^{96} using 2^{42} packets. So far, this is the best temporal key recovery attack against WPA.

We have shown that conditional biases are not very helpful for breaking WPA, but they really are against WEP. For WEP, we recover the secret key with a success rate of 50% by using 19 800 packets in less than a minute using a sequential distinguishing approach. This improve the attack of Beck and Tews [BT09] using 30 000 packets. The attack is still feasible with less number of packets, but it runs for a longer period.

Further Work. We did not perform any concrete theoretical analysis of the sequential distinguishing approach in the WEP attack. A further work could be to perform a precise analysis of this approach and compare it with the "interactive mode" in Aircrack-ng. We also plan to study further key recovery attacks on WPA to recover more pieces of the TK with a complexity lower than 2^{96}.

Part II

Algebraic Cryptanalysis of A Few Symmetric Cryptosystems

Chapter 10

ElimLin Algorithm for Solving Polynomial Systems of Equations

10.1 Introduction

Various techniques exist in cryptanalysis of symmetric ciphers. Some involve statistical analysis and some are purely deterministic. One of the latter methods is *algebraic attack* formulated as early as 1949 by Shannon [Sha49].

Any algebraic attack consists of two distinct stages:

- Writing the cipher as a system of polynomial equations of low degree often over $\mathsf{GF}(2)$ or $\mathsf{GF}(2^k)$, which is feasible for any cipher [Wei03, CP02, MR02].
- Recovering the secret key by solving such a large system of polynomial equations.

Algebraic attacks have been successful in breaking several stream ciphers (see [AA09, CM03, Cou03, DS11, COQ09, Cou09, DS09, Cou02] for instance) and a few block ciphers, for instance Keeloq [IKD+08] and GOST [Cou11], but they are not often as successful as statistical attacks. On the other hand, they often require low data complexity.

General purpose algebraic attack techniques were developed in the last few years by Courtois, Bard, Meier, Faugère, Raddum, Semaev, Vielhaber, Dinur and Shamir to solve these systems [CB07, CSPK00, CP02, CM03, Cou03, Fau99, Fau02, RS08, Vie07, DS09, DS11]. The problem of solving such polynomial systems of multivariate equations is called the MQ problem which is known to be NP hard in general. Currently, for a random system in which the number of equations is equal to the number of unknowns, there exists no technique faster than exhaustive key search to solve such systems. On the other hand, the equations derived from symmetric ciphers turn out to be overdefined and sparse for most ciphers. So, they are easier to solve. This sparsity is coming from the fact that, due to the limitations in hardware and the need for lightweight algorithms, simple operations arise in the definition of cryptosystems. They are also overdefined due to the non-linear operations.

The traditional methods for solving overdefined polynomial systems of equations are known to be various Gröbner basis algorithms such as Buchberger algorithm [Buc06], F4 and F5 [Fau99, Fau02] and XL [CSPK00]. The most critical drawback of the Gröbner basis approach is the elimination step where the degree of the system increases. This leads to an explosion in memory space and in the worst case scenario, they run in double exponential time, and even the most efficient implementations of Faugère algorithms [Fau99, Fau02] under PolyBoRi framework [BD07] or Magma [Mag] are not capable of handling *large* systems of equations efficiently. On the other hand, they are faster than other methods for overdefined dense systems or when the equations are over GF(q) where $q > 2$. In fact, together with SAT solvers, they are currently the most successful methods for solving polynomial systems.

Nevertheless, due to the above mentioned technical reasons, the system of equations extracted from symmetric ciphers turns out to be sparse. Unfortunately, the Gröbner basis algorithms can not exploit this property. In such cases, algorithms such as XSL [CP02], SAT solving techniques [Bar09, ES, BCJ07], Raddum-Semaev algorithm [RS08] and ElimLin [CB07] are of interest.

In this chapter, we study the elimination algorithm ElimLin that falls within the remit of Gröbner basis algorithms, though it is conceptually much simpler and is based on a mix of simple linear algebra and substitution. It maintains the degree of the equations and it does not require any fixed ordering on the set of all monomials. This is not the case for the Gröbner basis algorithms, where monomial ordering is a prominent factor. On the contrary, we need to work with ad-hoc monomial orderings to preserve the sparsity and make it run faster. This simple algorithm reveals some hidden linear equations existing in the ideal generated by the system. We show in Sec. 10.4.4 that ElimLin does not find all such linear equations.

As far as we are aware, no clue has been yet found which demonstrates that ElimLin power is limited. This does not mean that ElimLin can break any system. As mentioned earlier, MQ problem is NP hard and Gröbner basis algorithms behave much better for such dense random systems. But, the equations derived from cryptosystems are often not random (see [FB09] for the huge difference between a random system and the algebraic representation of cryptographic protocols). What we mean here is that, if for some small number of rounds ElimLin performs well but then it fails for more rounds, we can increase the number of samples and it will become effective again. The bottleneck is having an efficient data structure for implementing ElimLin together with a rigorous theory behind it to anticipate its behaviour.

Except for two simple theorems by Bard (see Chapter 12, Sec. 5 of [Bar09]), almost nothing has been done regarding the theory behind ElimLin. As ElimLin can also be used as a pre-processing step in any algebraic attack, building a proper theory is vital for improving the state of the art algebraic attacks. We are going to shed some light on the way this ad-hoc algorithm works and the theory behind it.

In this chapter, we show that the output of ElimLin is invariant with respect to any variable ordering. This is a surprising result, i.e., while the spaces generated are different depending on how substitution is performed, we prove that their intersection is exactly the same. Furthermore, we prove that no affine bijective variable change can modify the output of ElimLin.

In Sec. 10.2, we elaborate the ElimLin algorithm. Then, in Sec. 10.2.1, we give a toy example on ElimLin. We remind some basic theorems on ElimLin in Sec. 10.3. As our main contribution (Theorem 10.2), we prove in Sec. 10.4 that ElimLin can be formulated as an intersection of vector spaces. We also discuss its consequences in Sec. 10.4.2 and remind a theorem regarding the evolution of linear equations in Sec. 10.4.3. We perform some attack simulations on LBlock and

PRESENT block ciphers in Sec. 10.5. In Sec. 10.6, we compare ElimLin and F4.

10.2 ElimLin Algorithm

ElimLin stands for **Elim**inate **Lin**ear and it is a technique for solving polynomial systems of multivariate equations of low degree d mostly: 2, 3, or 4 over a finite field specifically GF(2). It is also known as an "inter-reduction" step in all major algebra systems. It was proposed as a single tool in [CB07] to attack DES. It broke 5-round DES. Later, it was applied to analyze the resistance of Snow 2.0 stream cipher against algebraic attacks [CD08]. It is a simple but a powerful algorithm which can be applied to any symmetric cipher and is capable of breaking their reduced versions. There is no specific requirement for the system, except that there should exist at least one degree one term, otherwise ElimLin trivially fails. The key question for such an algorithm is to predict its behavior. Currently, very similar to most other types of algebraic attacks such as [Vie07, DS09, DS11], multiple parts of the algorithm are heuristic, so it is worthwhile to prove which factors can improve its results, i.e., makes it generate more linear equations, make it run faster or does not have any influence on its ultimate result. This will yield a better understanding of how ElimLin works.

ElimLin is composed of two sequential distinct stages, namely:

- *Gaussian elimination:* All the linear equations in the linear span of the initial equations are found. They are the intersection between two vector spaces: The vector space spanned by all monomials of degree 1 and the vector space spanned by all equations.

- *Substitution:* Variables are iteratively eliminated in the whole system based on the linear equations until there is no linear equation left. Consequently, the remaining system has fewer variables.

This routine is iterated until no linear equation is obtained in the linear span of the system. See Algorithm 10.1 for a more precise definition of the algorithm. We say that ElimLin *succeeds* if it can eliminate all the variables in the system, and it *fails* if it stops before doing so. If the system of equations represents the algebraic structure of a cipher, the success of ElimLin is equivalent to the cipher being broken. We also give a toy system of equations in the next section and solve it with ElimLin.

Clearly, the algorithm may depend on the ordering strategies to apply in step 5, 11 and 12 of Algorithm 10.1. We will see that it is not, i.e., the span of the resulting $\mathcal{S}_L$ is invariant.

We observe that new linear equations are derived in each iteration of the algorithm that did not exist in the former spans. This phenomenon is called *avalanche effect* in ElimLin and is the consequence of Theorem 10.2. At the end, the system is solved linearly (when $\mathcal{S}_L$ is large enough) or ElimLin fails. If the latter occurs, we can increase the data complexity (for instance, the number of plaintext-ciphertext pairs) and re-run the attack.

Algorithm 10.1 ElimLin algorithm.

Input : A system of polynomial equations $\mathcal{S}^0 = \{\mathsf{Eq}_1^0, \ldots, \mathsf{Eq}_{m_0}^0\}$ over GF(2).
Output : A system of linear equations $\mathcal{S}_L$.
Set $\mathcal{S}_L \leftarrow \emptyset$ and $\mathcal{S}^T \leftarrow \mathcal{S}^0$ and $k \leftarrow 1$.
repeat
  Perform Gaussian elimination Gauss(.) on $\mathcal{S}^T$ with an arbitrary ordering of equations and monomials to eliminate non-linear monomials.
  Set $\mathcal{S}_{L'} \leftarrow$ Linear equations from Gauss($\mathcal{S}^T$).
  Set $\mathcal{S}^T \leftarrow \mathsf{Gauss}(\mathcal{S}^T) \setminus \mathcal{S}_{L'}$.
  Set flag.
  for all $\ell \in \mathcal{S}_{L'}$ in an arbitrary order **do**
    if ℓ is a trivial equation **then**
      if ℓ is unsolvable **then**
        Terminate and output "No Solution".
      end if
    else
      Unset flag.
      Let x_{t_k} be a monomial from ℓ.
      Substitute x_{t_k} in $\mathcal{S}^T$ and $\mathcal{S}'_L$ using ℓ.
      Insert ℓ in $\mathcal{S}_L$.
      $k \leftarrow k + 1$
    end if
  end for
until flag is set.
Output $\mathcal{S}_L$.

10.2.1 A Toy Example of ElimLin

Let's assume that we have the following overdefined system of multivariate equations over GF(2) with 5 variables $x_1, \ldots, x_5$ and 6 equations:

$$\begin{cases} x_1x_2 + x_1x_3 + x_2x_5 + x_3x_5 + x_2 + x_4 + x_5 + 1 = 0 \\ x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 = 0 \\ x_1x_4 + x_2x_3 + x_3 + 1 = 0 \\ x_1x_4 + x_1x_5 + x_2x_5 + x_1 = 0 \\ x_1x_5 + x_2x_3 + x_3x_5 + x_1 + x_3 + x_4 = 0 \\ x_1x_5 + x_2x_3 + x_3x_5 + x_5 + x_4 + x_2 + 1 = 0 \end{cases}$$

We perform Gaussian elimination on the system, and obtain:

$$\begin{cases} x_1x_2 + x_2x_4 + x_2x_5 + x_3x_5 + x_2 + x_3 + x_4 + x_5 = 0 \\ x_1x_3 + x_2x_4 + x_3 + 1 = 0 \\ x_1x_4 + x_2x_3 + x_3 + 1 = 0 \\ x_1x_5 + x_2x_3 + x_3x_5 + x_1 + x_3 + x_4 = 0 \\ x_2x_5 + x_3x_5 + x_4 + 1 = 0 \\ x_1 + x_2 + x_3 + x_5 + 1 = 0 \end{cases}$$

The linear equation we obtain is used for the substitution of the variable $x_5 = x_1 + x_2 + x_3 + 1$. Then, we perform Gaussian elimination on the system again. We derive:

$$\begin{cases} x_2x_4 + x_1 = 0 \\ x_1x_4 + x_2x_3 + x_3 + 1 = 0 \\ x_1x_2 + x_1 + x_3 + x_4 = 0 \\ x_1x_3 + x_1 + x_3 + 1 = 0 \\ x_4 + x_3 + x_1 = 0 \end{cases}$$

The new linear equation is used for the substitution of the variable $x_4 = x_3 + x_1$. After the substitution, we perform Gaussian elimination again and obtain:

$$\begin{cases} x_2x_3 + x_1 = 0 \\ x_1x_3 + x_3 + 1 = 0 \\ x_1x_2 = 0 \\ x_1 = 0 \end{cases}$$

We derive a new linear equation $x_1 = 0$. Consequently, we perform a substitution and Gaussian elimination, which yields:

$$\begin{cases} x_2x_3 = 0 \\ x_3 + 1 = 0 \end{cases}$$

The new linear equation we obtain is $x_3 = 1$. After the substitution of this variable, we obtain $x_2 = 0$. Hence, we have gathered 5 linear equations in 5 variables as follows, which can be simply solved by:

$$\begin{cases} x_1 + x_2 + x_3 + x_5 + 1 = 0 \\ x_1 + x_3 + x_4 = 0 \\ x_1 = 0 \\ x_3 + 1 = 0 \\ x_2 = 0 \end{cases}$$

leading to $x_1 = x_2 = x_5 = 0$ and $x_3 = x_4 = 1$.

10.2.2 Optimization

ElimLin running time is highly dependent on the method to achieve the row echelon form and how to perform the substitution. It is necessary to use ad-hoc strategies to preserve the sparsity. Although the algorithm is very simple, an efficient implementation of it is hard to obtain and we used various heuristics in our implementation to preserve the sparsity and to make the algorithm fast, like the choice of the leading variable to eliminate and the method to perform the Gaussian reduction.

Gaussian elimination can be performed in various ways. What is often done is to pick a monomial ordering for equations and eliminate the leading monomial of the active row. This approach does not have any effect on preserving the sparsity. Consequently, we mention several sparsity preserving heuristics:

– create a list of monomials, together with the number of times they have occurred in the system and try to eliminate those which are repeated less first. This will preserve the sparsity and speculated to make the Gaussian reduction process faster. In this approach, as the intermediate key variables are those which are repeated the most, they will be eliminated after all other intermediate variables are omitted. Therefore, gathering some *"statistics"* is required in the implementation phase. Moreover, the number of times the statistics should be updated and a method for an effective implementation affect the speed of ElimLin dramatically.

– another strategy is to eliminate the equation which is smaller in size called *pivot equation* first. This technique is referred to as *"naive Gaussian elimination"* or *"structured Gaussian elimination"* [Bar09]. It has effectively been used in SAGE [Ste] algebra system for sparse matrix elimination over GF(2).

– the two previous methods can be merged: take an equation with a monomial which repeats less and if two monomials are repeated the same number of times, eliminate the equation that is smaller in weight. It turned out that this approach is the fastest in our implementation of ElimLin algorithm.

In the substitution layer, if the weight of the linear equation to be substituted is smaller, it is more probable that the sparsity is preserved. Hence, the basis acquired by performing ElimLin may not be the most appropriate required for substitution.

– An effective, but expensive algorithm using *"Gray code"* and *"rapid subspace enumeration"* was proposed by Bard in [Bar09] for constructing a low weight basis.

Moreover, substitution can be performed in different ways. Our implementation showed that the direct method is quite slow. We found a better representation of the substitution stage which is represented in Lemma 10.1.

10.3 State of the Art Theorems

The only theoretical analysis of ElimLin was done by Bard in [Bar09]. He proved the following theorem and corollary for **one** iteration of ElimLin:

Theorem 10.1 ([Bar09])**.** *All linear equations in the linear span of a polynomial equation system $\mathcal{S}^0$ are found in the linear span of linear equations derived by performing the first iteration of ElimLin algorithm on the system.*

The following corollary (also from [Bar09]) is the direct consequence of the above theorem.

Corollary 10.1. *The linear equations generated after performing the first Gaussian elimination in ElimLin algorithm form a basis for all possible linear equations in the linear span of the system.*

This shows that any method to perform Gaussian elimination does not affect the linear space obtained at an arbitrary iteration of ElimLin. All linear equations derived from one method exist in the linear span of the equations cumulated from another method. This is trivial to see.

10.4 Algebraic Representation of ElimLin

10.4.1 ElimLin as an Intersection of Vector Spaces

We also formalize ElimLin in an algebraic way. This representation is used in proving Theorem 10.2. Firstly, we define some notations.

We call an *iteration* a Gaussian elimination preceding a substitution; The system of equations for ElimLin can be stored as a matrix $\mathcal{M}_\alpha$ of dimension $m_\alpha \times T_\alpha$, where each m_α rows represents an equation and each T_α columns represents a monomial at iteration α. Also, r_α denotes the rank of $\mathcal{M}_\alpha$. Let n_α be the number of variables at iteration α. We use a reverse lexicographical ordering of columns during Gaussian elimination to accumulate linear equations in the last rows of the matrix. Any arbitrary ordering can be used instead. In fact, we use the same matrix representation as described in [Bar09].

Let $K = \mathsf{GF}(2)$ and $x = (x_1, \ldots, x_n)$ be a set of variables. We denote by $K[x]$ the ring of multivariate polynomials over K. For $\mathcal{S} \subset K[x]$, we denote $\mathsf{Span}(\mathcal{S})$ the K-vector subspace of $K[x]$ spanned by $\mathcal{S}$. Let $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n)$ be a vector in $\mathbf{N}^n$ and γ is referred to as a *"power vector"*. The term x^γ is defined as the product $x^\gamma = x_1^{\gamma_1} \times x_2^{\gamma_2} \times \cdots \times x_n^{\gamma_n}$. The total degree of x^γ is defined as $deg(x^\gamma) \stackrel{\mathsf{def}}{=} \gamma_1 + \gamma_2 + \cdots + \gamma_n$. Let $\mathsf{Ideal}(\mathcal{S})$ be the ideal spanned by $\mathcal{S}$ and $\mathsf{Root}(\mathcal{S})$ be the set of all tuples $m \in K^n$ such that $f(m) = 0$ for all $f \in \mathcal{S}$. Let

$$R_d = \mathsf{Span}\left(\text{monomials of degree} \leq d\right) / \mathsf{Ideal}\left(x_1^2 - x_1, x_2^2 - x_2, \ldots, x_n^2 - x_n\right)$$

Let $\mathcal{S}^\alpha$ be $\mathcal{S}^T$ after the α-th iteration of ElimLin and $\mathcal{S}^0$ be the initial system. Moreover, n_L^α is the number of non-trivial linear equations in $\mathcal{S}_{L'}$ at the α-th iteration. Trivial equations are equations in the form $0 = 0$ or $0 = 1$. We denote $\mathcal{S}_L^\alpha$ the $\mathcal{S}_L$ after the α-th iteration. Also,

$$C^\alpha \stackrel{\mathsf{def}}{=} \#\mathcal{S}_L^\alpha$$

Let's assume that $\mathcal{S}^0$ has degree bounded by d. We denote by $\mathsf{Var}(f)$ the set of variables x_i occurring in f. Let $x_{t_1}, \ldots, x_{t_k}$ be the sequence of eliminated variables. We define $\mathsf{V}_k = \{x_1, \ldots, x_n\} \backslash \{x_{t_1}, \ldots, x_{t_k}\}$. Also, let $\ell_1, \ell_2, \ldots, \ell_k$ be the sequence of linear equations as they are used during elimination (step 16 of Algorithm 10.1). Hence, we have $x_{t_k} \in \mathsf{Var}(\ell_k) \subseteq \mathsf{V}_{k-1}$.

We prove the following crucial lemma which we use later to prove Theorem 10.2.

Lemma 10.1. *After the α-th iteration of ElimLin, an arbitrary equation Eq_i^α in the system $(\mathcal{S}^\alpha \cup \mathcal{S}_L^\alpha)$ for an arbitrary i can be represented as*

$$\mathsf{Eq}_i^\alpha = \sum_{t=1}^{m_0} \beta_{ti}^\alpha \cdot \mathsf{Eq}_t^0 + \sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g_{ti}^\alpha(x) \tag{10.1}$$

where $\beta_{ti}^\alpha \in K$ and $g_{ti}^\alpha(x)$ is a polynomial in R_{d-1} and $\mathsf{Var}(g_{ti}^\alpha) \subseteq \mathsf{V}_t$.

Proof. Let x_{t_1} be one of the monomials existing in the first linear equation $\ell_1(x)$ and this specific variable is going to be eliminated. Substituting x_{t_1} in an equation $x_{t_1} \cdot h(x) + z(x)$, where $h(x)$ has degree at most $d-1$, $x_{t_1} \notin \mathsf{Var}(h)$ and $x_{t_1} \notin \mathsf{Var}(z)$ is identical to subtracting $h(x) \cdot \ell_1(x)$. Consequently, the proof follows by induction on α. □

Now, we prove the inverse of the above lemma.

Lemma 10.2. *For each i and each α, there exists $\beta'^{\alpha}_{ti} \in K$ and $g'^{\alpha}_{ti}(x)$ such that*

$$\mathsf{Eq}_i^0 = \sum_{t=1}^{m_\alpha} \beta'^{\alpha}_{ti} \cdot \mathsf{Eq}_t^\alpha + \sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g'^{\alpha}_{ti}(x) \tag{10.2}$$

where $g'^{\alpha}_{ti}(x)$ is a polynomial in R_{d-1} and $\mathsf{Var}(g'^{\alpha}_{ti}) \subseteq \mathsf{V}_t$.

Proof. Gaussian elimination and substitution are invertible operations. We can use a similar induction as the previous lemma to prove the above equation. □

In the next lemma, we prove that $\mathcal{S}_L^\alpha$ contains all linear equations which can be written in the form of Eq. (10.1).

Lemma 10.3. *If there exists $\ell \in R_1$ and some β_t and $g''_t(x)$ such that*

$$\ell(x) = \sum_{t=1}^{m_0} \beta_t \cdot \mathsf{Eq}_t^0 + \sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g''_t(x) \tag{10.3}$$

at iteration α, where $g''_t(x)$ is a polynomial in R_{d-1}, then there exist $u_t \in K$ and $v_t \in K$ such that

$$\ell(x) + \sum_{t=1}^{C^\alpha} u_t \cdot \ell_t(x) = \sum_{t=1}^{m_\alpha} v_t \cdot \mathsf{Eq}_t^\alpha$$

So, $\ell(x) \in \mathit{Span}(\mathcal{S}_L^\alpha)$.

Proof. We define u_k iteratively: u_k is the coefficient of x_{t_k} in

$$\ell(x) + \sum_{t=1}^{k-1} u_t \cdot \ell_t(x)$$

for $k = 1, \ldots, C^\alpha$. So, $\mathsf{Var}(\ell(x) + \sum_{t=1}^{k} u_t \cdot \ell_t(x)) \subseteq \mathsf{V}_k$. By substituting Eq_i^0 from Eq. (10.2) in Eq. (10.3) and substituting u_t and g''_t in g'^{α}_{ti}, we obtain

$$\underbrace{\ell(x) + \sum_{t=1}^{C^\alpha} u_t \cdot \ell_t(x)}_{\subseteq \mathsf{V}_1} = \underbrace{\sum_{t=1}^{m_\alpha} v_t \cdot \mathsf{Eq}_t^\alpha}_{\subseteq \mathsf{V}_1} + \underbrace{\sum_{t=1}^{C^\alpha} \ell_t(x) \cdot g'_t(x)}_{\Longrightarrow \subseteq \mathsf{V}_1} \tag{10.4}$$

with $g'_t(x) \in R_{d-1}$. All $g'_t(x)$ where $t > 1$ can be written as $\bar{g}_t(x) + x_{t_1} \cdot \bar{\bar{g}}_t(x)$ with $\mathsf{Var}(\bar{g}_t) \subseteq \mathsf{V}_1$, $\mathsf{Var}(\bar{\bar{g}}_t) \subseteq \mathsf{V}_1$ and $\bar{\bar{g}}_t(x) \in R_{d-2}$. As,

$$\ell_1(x) \cdot g'_1(x) + \ell_t(x) \cdot g'_t(x) = \ell_1(x) \cdot \underbrace{\big(g'_1(x) + \ell_t(x) \cdot \bar{\bar{g}}_t(x)\big)}_{\text{new } g'_1(x)} + \underbrace{\ell_t(x)}_{\subseteq \mathsf{V}_1} \cdot \underbrace{\big(\bar{g}_t(x) + \bar{\bar{g}}_t(x) \cdot (x_{t_1} - \ell_1(x))\big)}_{(\text{new } g'_t(x)) \subseteq \mathsf{V}_1}$$

we can re-arrange the sum in Eq. (10.4) using the above representation and obtain $\mathsf{Var}(g'_t) \subseteq \mathsf{V}_1$ for all $t > 1$. Also, x_{t_1} only appears in $\ell_1(x)$ and $g'_1(x)$. So, the coefficient of x_{t_1} in the expansion of $\ell_1(x) \cdot g'_1(x)$ must be zero. In fact, we have

$$\begin{aligned} \ell_1(x) \cdot g'_1(x) &= (x_{t_1} + (\ell_1(x) - x_{t_1})) \cdot (\bar{g}_1(x) + x_{t_1} \cdot \bar{\bar{g}}_1(x)) \\ &= x_{t_1} \cdot (\bar{\bar{g}}_1(x) \cdot (1 + \ell_1(x) - x_{t_1}) + \bar{g}_1(x)) + \bar{g}_1(x) \cdot (\ell_1(x) - x_{t_1}) \end{aligned}$$

So, $\bar{g}_1(x) = \bar{\bar{g}}_1(x) \cdot (x_{t_1} - \ell_1(x) - 1)$ and we deduce,

$$g_1'(x) = \bar{\bar{g}}_1(x) \cdot (\ell_1(x) + 1)$$

over $\mathsf{GF}(2)$. But, then

$$\ell_1(x) \cdot g_1'(x) = 0$$

over R, since $\ell_1(x) \cdot (\ell_1(x) + 1) = 0$. Finally, we iterate and obtain

$$\ell(x) + \sum_{t=1}^{C^\alpha} u_t \cdot \ell_t(x) = \sum_{t=1}^{m_\alpha} v_t \cdot \mathsf{Eq}_t^\alpha$$

□

From another perspective, ElimLin algorithm can be represented as in Algorithm 10.2. In fact, as a consequence of Lemma 10.1 and Lemma 10.3, Algorithm 10.2 presents a unique characterization of $\mathsf{Span}\,(\mathcal{S}_L)$ in terms of a fixed point:

Algorithm 10.2 ElimLin algorithm from another perspective.

1: Input : A set $\mathcal{S}^0$ of polynomial equations in R_d.
2: Output : A system of linear equations $\mathcal{S}_L$.
3: Set $\bar{\mathcal{S}_L} := \emptyset$.
4: **repeat**
5: $\quad \bar{\mathcal{S}_L} \leftarrow \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$
6: **until** $\bar{\mathcal{S}_L}$ unchanged
7: Output $\mathcal{S}_L$: a basis of $\bar{\mathcal{S}_L}$.

Lemma 10.4. *At the end of ElimLin, $\mathsf{Span}\,(\mathcal{S}_L)$ is the smallest subset $\bar{\mathcal{S}_L}$ of R_1, such that*

$$\bar{\mathcal{S}_L} = \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$$

Proof. By induction, at step α we have $\bar{\mathcal{S}_L} \subseteq \mathsf{Span}\,(\mathcal{S}_L^\alpha)$, using Lemma 10.3. Also, $\mathcal{S}_L^\alpha \subseteq \bar{\mathcal{S}_L}$ using Lemma 10.1. So, $\bar{\mathcal{S}_L} = \mathsf{Span}\,(\mathcal{S}_L^\alpha)$ at step α. Since $\bar{\mathcal{S}_L} \mapsto \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1$ is increasing, we obtain the above equation. □

ElimLin eliminates variables, thus it looks very unexpected that the number of linear equations in each step of the algorithm is invariant with respect to any variable ordering in the substitution step and the Gaussian elimination. We finally prove this important invariant property. Concretely, we formalize ElimLin as a sequence of intersection of vector spaces. Such intersection in each iteration is between the vector space spanned by the equations and the vector space generated by all monomials of degree 1 in the system. This implies that if ElimLin runs for α iterations (finally succeeds or fails), it can be formalized as a sequence of intersections of α pairs of vector spaces. The final sequence of intersections of these vector spaces only depends on the vector space of the initial system.

Theorem 10.2. *The following relations exist after running ElimLin on a polynomial system of equations Q:*

1. $\mathsf{Root}\mathcal{S}^0 = \mathsf{Root}(\mathcal{S}^T \cup \mathcal{S}_L)$

2. *There is no linear equation in* $\mathsf{Span}\left(\mathcal{S}^T\right)$.

3. $\mathsf{Span}\left(\mathcal{S}_L\right)$ *is uniquely defined by* $\mathcal{S}^0$.

4. $\mathcal{S}_L$ *consists of linearly independent linear equations.*

5. *The complexity of* ElimLin *is* $O\left(n_0^{d+1}m_0^2\right)$, *where* d *is the degree of the system and* n_0 *and* m_0 *are the initial number of variables and equations, respectively.*

Proof. *1.* Due to Lemma 10.1 and Lemma 10.2, $\mathcal{S}^0$ and $(\mathcal{S}^T \cup \mathcal{S}_L)$ are equivalent. In fact, a solution of $\mathcal{S}^0$ is also a solution of $(\mathcal{S}^T \cup \mathcal{S}_L)$ and vice versa.

2. Since ElimLin stops on $\mathcal{S}^T$, the Gaussian reduction did not find any linear polynomial.

3. Due to Lemma 10.4.

4. $\mathcal{S}_L$ is a basis for $\bar{\mathcal{S}}_L$. So, it consists of linearly independent equations.

5. n_0 is an upper bound on $\#\mathcal{S}_L$ due to the fact that $\mathcal{S}_L$ consists of linearly independent linear equations. So, the number of iterations is bounded by n_0. The total number of monomials is bounded by

$$T_0 \leq \sum_{i=0}^{d} \binom{n_0}{i} = O\left(n_0^d\right)$$

The complexity of Gaussian elimination is $O(m_0^2 T_0)$, as we have T_0 columns and m_0 equations. Therefore, overall the complexity of ElimLin is $O\left(n_0^{d+1}m_0^2\right)$. □

10.4.2 Affine Bijective Variable Change

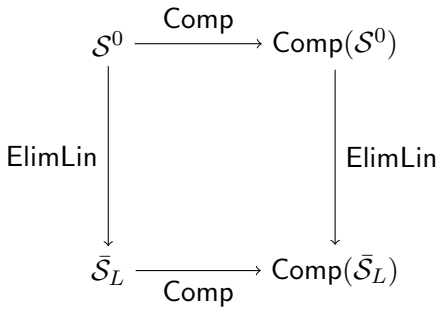
In the next theorem, we prove that the result of ElimLin algorithm does not change for any affine bijective variable change.

Theorem 10.3. *Any affine bijective variable change* $A : \mathsf{GF}(2)^{n_0} \rightarrow \mathsf{GF}(2)^{n_0}$ *on a* n_0*-variable system of equations* $\mathcal{S}^0$ *does not affect the result of* ElimLin *algorithm, implying that the number of linear equations generated at each iteration is invariant with respect to an affine bijective variable change.*

Proof. In Lemma 10.4, we showed that $\mathsf{Span}\left(\mathcal{S}_L\right)$ is the output of the algorithm in Algorithm 10.2, iterating

$$\bar{\mathcal{S}}_L \leftarrow \mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}}_L)\right) \cap R_1$$

We represent the composition of a polynomial f_1 with respect to A by $\mathsf{Comp}(f_1)$. We then show that there is a commutative diagram



We consider two parallel executions of the algorithm in Algorithm 10.2, one with $\mathcal{S}^0$ and the other with $\mathsf{Comp}(\mathcal{S}^0)$. If we compose the polynomials in $\mathcal{S}^0$ with respect to A, in the above relation R_{d-1} remains the same. As the transformation A is affine,

$$\mathsf{Comp}(\mathsf{Span}\left(\mathcal{S}^0 \cup (R_{d-1} \times \bar{\mathcal{S}_L})\right) \cap R_1) = \mathsf{Span}\left(\mathsf{Comp}(\mathcal{S}^0) \cup (R_{d-1} \times \mathsf{Comp}(\bar{\mathcal{S}_L}))\right) \cap R_1$$

So, at each iteration, the second execution has the result of applying Comp to the result of the first one. □

10.4.3 Linear Equations Evolution

An open problem regarding ElimLin is to predict how the number of linear equations evolves in the preceding iterations. The following theorem gives a necessary (but not sufficient) condition for a dense overdefined system of equations to have an additional linear equation in the next iteration of ElimLin. See [SSV12] or [Sep08] for the proof, proving a similar result for a sparse system is not straightforward.

Theorem 10.4. *If we apply ElimLin to an overdefined dense system of quadratic equations over* GF(2)*, for* $n_L^{\alpha+1} > n_L^{\alpha}$ *to hold, it is necessary to have*

$$\frac{b_\alpha}{2} - a_\alpha < n_L^\alpha < \frac{b_\alpha}{2} + a_\alpha$$

where $b_\alpha = 2n_\alpha - 1$ *and* $a_\alpha = \frac{\sqrt{b_\alpha^2 - 8n_L^\alpha}}{2}$.

10.4.4 Does ElimLin Find All the Linear Equations?

We give an example that illustrates that elimlin does not reveal all hidden linear equations in the structure of the cipher up to a specific degree:

Assume that there exists an equation in the system which can be represented as $\ell(x)g(x)+1 = 0$ over GF(2), where $\ell(x)$ is a polynomial of degree one and $g(x)$ is a polynomial of degree at most $d-1$. Running ElimLin on this single equation trivially fails. But, if we multiply both sides of the equation by $\ell(x)$, we obtain $\ell(x) \cdot g(x) + \ell(x) = 0$. Summing these two equations, we derive $\ell(x) = 1$. Indeed, $(\ell(x) \cdot g(x) + 1)(\ell(x) + 1) = \ell(x) + 1$. This hidden linear equation can be simply captured by the XL algorithm, but can not be captured by ElimLin. There exist multiple other examples which demonstrate that ElimLin does not generate all the hidden linear equations. To generate all such linear equations, the degree-bounded Gröbner basis can be used.

10.5 Attack Simulations

In this section, we present our experimental results against LBlock and PRESENT block ciphers using ElimLin and F4 algorithms. All the simulations were run on an ordinary PC with a (2.8 Ghz CPU with 4 GB RAM) and a (2.0 Ghz CPU with 1 GB RAM) respectively.

In our attacks, we build a system of quadratic equations with variables representing the plaintext, the ciphertext, the key and the state bits, which allows to express the system of equations of high degree as quadratic equations. Afterwards, for each sample we set the plaintext and ciphertext according to the result of the input/output of the cipher. In order to test the efficiency of the algebraic attack, we guess some bits of the key and set the key variables corresponding to the guess. Then, we run the solver (ElimLin or F4) to recover the remaining key bits and test whether the guess was correct. Therefore, the complexity of our algebraic attack can be bounded by $2^g \cdot \mathcal{C}(solver)$, where $\mathcal{C}(solver)$ represents the running time of the solver and g is the number of bits we guess. $\mathcal{C}(solver)$ is represented as the *"Running Time"* in Table 10.1.

For a comparison with a brute force attack, we consider a fair implementation of the cipher, which requires 10 CPU cycles per round. This implies that the algebraic attack against t rounds of the cipher is faster than exhaustive search for the 2.8 Ghz CPU (resp. 2.0 Ghz CPU) iff recovering c bits of the key is faster than $3.57t \cdot 2^{c-31}$ (resp. $5t \cdot 2^{c-31}$) seconds. This is already two times faster than the complexity of exhaustive search. The attack reported in Table 10.1 is faster than exhaustive search with the former argument. In fact, we consider the cipher to be broken for some number of rounds if the algebraic attack that recovers $(\#key - g)$ key bits is faster than exhaustive key search over $(\#key - g)$ bits of the key.

10.5.1 Simulations Using F4 Algorithm under PolyBoRi Framework

The most efficient implementation of the F4 algorithm is available under PolyBoRi framework [BD07] running alone or under SAGE algebra system. PolyBoRi is a C++ library designed to compute Gröbner basis of an ideal applied to Boolean polynomials. A Python interface is used, surrounding the C++ core. It uses zero-suppressed binary decision diagrams (ZDDs) [Gha05] as a high level data structure for storing Boolean polynomials. This representation stores the monomials more efficiently in memory and it performs the Gröbner basis computation faster compared to other algebra systems.

We use polybori-0.8.0 for our attacks. Together with ElimLin, we also attack LBlock with F4 algorithm and then compare its efficiency with ElimLin.

10.5.2 Attacking LBlock with ElimLin and F4

LBlock is a new lightweight Feistel-based block cipher proposed at ACNS 2011 [WZ11], aimed at constrained environments, such as RFID tags and sensor networks. It operates on 64-bit blocks, uses a key of 80 bits and iterates 32 rounds.

The encryption function of LBlock is illustrated in Fig. 10.1. Let $M = X_1||X_0$ denote a 64-bit plaintext, where each X_i is 32 bits. The data processing function can be expressed as in Algorithm 10.3, where $[K_1, \ldots, K_{32}]$ are thirty two 32-bit round keys and $<<< n$ represents an

n-bit left cyclic shift operation.

Algorithm 10.3 The encryption algorithm of LBlock.

1: Input: $M = X_1||X_0$ and $[K_1, \ldots, K_{32}]$
2: Output: $C = \text{LBlock}(M, [K_1, \ldots, K_{32}])$
3: **for** $i = 2$ to 33 **do**
4: $\quad X_i = F(X_{i-1}, K_{i-1}) \oplus (X_{i-2} <<< 8)$
5: **end for**
6: Output $C = X_{32}||X_{33}$.

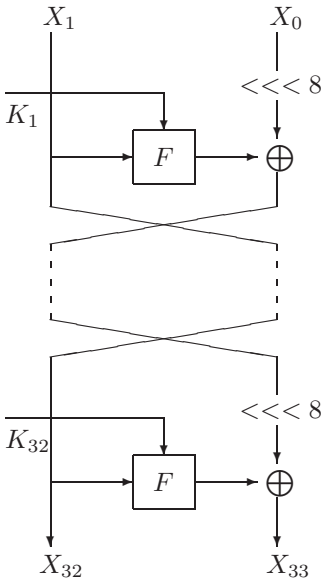


Figure 10.1: The encryption function of LBlock.

The round function F is defined as follows, where S and P denote the confusion and the diffusion functions. The S (S-box) non-linear functions are depicted in Figure 10.2. The last two S-boxes will be used in the key schedule.

$$\begin{aligned} F : \{0,1\}^{32} \times \{0,1\}^{32} &\rightarrow \{0,1\}^{32} \\ (X, K_i) &\rightarrow U = P(\mathsf{S}(X + K_i)) \end{aligned}$$

The F function is sketched in Figure 10.3.

The 80-bit master key K is stored in a key register and is denoted as $K = k_{79}k_{78}\ldots k_0$. To generate the round keys, LBlock goes through the Algorithm 10.4.

We break 8 rounds of LBlock using 6 samples deploying an ordinary PC by ElimLin. Our results are summarized in Table 10.1. In the same scenario, PolyBoRi crashes due to running out of memory.

Our goal is to show that multiple instances of a system of equations using distinct samples are in fact not independent. On one hand, having multiple instances of the system increases the

s_0	14, 9, 15, 0, 13, 4, 10, 11, 1, 2, 8, 3, 7, 6, 12, 5
s_1	4, 11, 14, 9, 15, 13, 0, 10, 7, 12, 5, 6, 2, 8, 1, 3
s_2	1, 14, 7, 12, 15, 13, 0, 6, 11, 5, 9, 3, 2, 4, 8, 10
s_3	7, 6, 8, 11, 0, 15, 3, 14, 9, 10, 12, 13, 5, 2, 4, 1
s_4	14, 5, 15, 0, 7, 2, 12, 13, 1, 8, 4, 9, 11, 10, 6, 3
s_5	2, 13, 11, 12, 15, 14, 0, 9, 7, 10, 6, 3, 1, 8, 4, 5
s_6	11, 9, 4, 14, 0, 15, 10, 13, 6, 12, 5, 7, 3, 8, 1, 2
s_7	13, 10, 15, 0, 14, 4, 9, 11, 2, 1, 8, 3, 7, 5, 12, 6
s_8	8, 7, 14, 5, 15, 13, 0, 6, 11, 12, 9, 10, 2, 4, 1, 3
s_9	11, 5, 15, 0, 7, 2, 9, 13, 4, 8, 1, 12, 14, 10, 3, 6

Figure 10.2: S-boxes of LBlock.

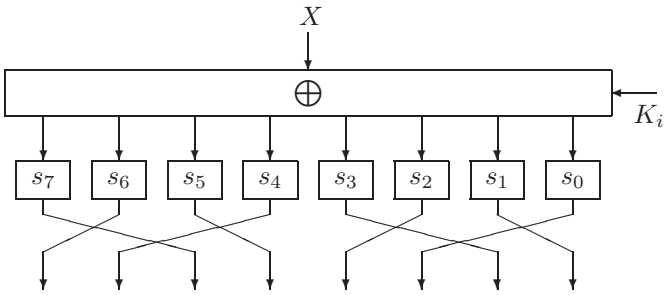


Figure 10.3: F function of LBlock.

number of variables, as the state bits are totally different, on the other hand, all the instances share the same key bits. The number of equations grows faster than the number of variables. Consequently, we expect that at one moment, the system totally collapses. We give an example using LBlock to be more clear. We attack 8-round LBlock with 32 LSB key bits fixed starting from 1 pair to 8 pairs. As can be observed from Table 10.2 to Table 10.9, the cipher is unbroken for 5 plaintext-ciphertext pairs, but then 6 pairs are enough to reveal enough linear equations to completely linearize the cipher, so that the system collapses.

For instance, in Table 11.7 we start with $n_0 = 8\,784$ variables and $m_0 = 27\,758$ equations. Here, T is the number of monomials. We then eliminate $n_L^1 = 7\,528$ variables at iteration 1. The variable elimination is repeated until the iteration 15, when we finish with 2 variables and $n_L^{15} = 2$ linear equations. As can be observed, at the last iteration the number of cumulative linear equations n_c is the same as the initial number of variables n_0. This implies that we only need to solve a linear system of equations in 8 784 variables and the system is solved. This is not the case for smaller number of samples. For instance, in Table 11.6, at the last iteration we finish with 499 variables and no linear equations. This implies that all 499 variables should be guessed.

In fact, for many cryptosystems, we observe that having a sufficient number of pairs is enough to completely linearize the system by ElimLin. As ElimLin is a polynomial time algorithm $\left(O(n_0^5)\right)$ in the initial number of variables, we believe that proving the former statement can already be considered as a breakthrough in cryptography.

Algorithm 10.4 The key scheduling algorithm of LBlock.

1: Input: $K = k_{79}k_{78}\dots k_0$
2: Output: $[K_1, \dots, K_{32}]$
3: Output the leftmost 32 bits of K as the first round key K_1.
4: **for** $i = 1$ to 31 **do**
5: update the register K as follows:

(a) $K <<< 29$

(b) $[k_{79}k_{78}k_{77}k_{76}] = s_9[k_{79}k_{78}k_{77}k_{76}]$

(c) $[k_{75}k_{74}k_{73}k_{72}] = s_8[k_{75}k_{74}k_{73}k_{72}]$

(d) $[k_{50}k_{49}k_{48}k_{47}k_{46}] \oplus [i]_2$

(e) Output the leftmost 32 bits of current content of K as the round key K_{i+1}.

6: **end for**

Table 10.1: Algebraic attack complexities on reduced-round LBlock using ElimLin and PolyBoRi.

N_r	$\#key$	g	Running Time (in hours)	Data	Attack notes
8	80	32	0.252	6 KP	ElimLin
8	80	32	crashed	6 KP	PolyBoRi

N_r : Number of rounds
g: Number of guessed LSB of the key
KP: Known plaintext
CP: Chosen plaintext

Table 10.2: Attacking 8-round LBlock with 1 pair and 32 LSB key bits guessed.

I	n	m_0	T	n_L	n_c
1	2064	5174	4249	1768	1768
2	296	5174	5678	42	1810
3	254	5174	5035	16	1826
4	238	5174	4868	3	1829
5	235	5174	5178	0	1829

Table 10.3: Attacking 8-round LBlock with 2 pairs and 32 LSB key bits guessed.

I	n	m_0	T	n_L	n_c
1	3408	8822	7385	2920	2920
2	488	8822	10855	85	3005
3	403	8822	11545	48	3053
4	355	8822	11955	22	3027
5	333	8822	13779	8	3035
6	325	8822	13729	0	3035

10.5.3 Attacking PRESENT with ElimLin and F4

PRESENT is an *Substitution Permutation Network* (SPN)-based block cipher designed for constrained environments, such as RFID tags and sensor networks. It was designed to be particularly compact and competitive in hardware. PRESENT operates on 64-bit text blocks, iterates 31 rounds and uses keys of either 80 or 128 bits. This cipher was designed by Bogdanov et al. and was released at CHES 2007 [BKL$^+$07]. Each (full) round of PRESENT contains three layers in the following order: a bitwise XOR layer with the round subkey; an S-box layer, in which a fixed

Table 10.4: Attacking 8-round LBlock with 3 pairs and 32 LSB key bits guessed.

I	n	m_0	T	n_L	n_c
1	4752	13110	10521	4072	4072
2	680	13110	16032	128	4200
3	552	13110	17495	83	4283
4	469	13110	18190	40	4323
5	429	13110	19913	21	4344
6	408	13110	20547	5	4349
7	403	13110	20843	1	4350
8	402	13110	20725	1	4351
9	401	13110	20561	0	4351

Table 10.5: Attacking 8-round LBlock with 4 pairs and 32 LSB key bits guessed.

I	n	m_0	T	n_L	n_c
1	6096	17839	13657	5224	5224
2	872	17839	21209	171	5395
3	701	17839	24035	118	5511
4	583	17839	25396	66	5577
5	517	17839	27955	40	5617
6	477	17839	31106	21	5638
7	456	17839	31611	16	5654
8	440	17839	28934	1	5655
9	439	17839	28717	0	5655

Table 10.6: Attacking 8-round LBlock with 5 pairs and 32 LSB key bits guessed.

I	n	m_0	T	n_L	n_c
1	7440	22730	16793	6376	6376
2	1064	22730	26386	214	6590
3	850	22730	30368	151	6741
4	699	22730	33097	91	6832
5	608	22730	37005	55	6887
6	553	22730	39058	35	6922
7	518	22730	37629	16	6938
8	502	22730	35748	1	6939
9	501	22730	35709	1	6940
10	500	22730	35509	1	6941
11	499	22730	34649	0	6941

Table 10.7: Attacking 8-round LBlock with 6 pairs and 32 LSB key bits guessed.

I	n	m_0	T	n_L	n_c
1	8784	27758	19929	7528	7528
2	1256	27758	31563	257	7785
3	999	27758	36607	189	7974
4	810	27758	41351	123	8097
5	687	27758	48066	83	8180
6	604	27758	46540	41	8221
7	563	27758	42910	15	8236
8	548	27758	41469	8	8244
9	540	27758	39312	24	8268
10	516	27758	29409	126	8394
11	390	27758	23370	108	8502
12	282	27758	14889	87	8589
13	195	27758	9157	122	8711
14	73	27758	1454	71	8782
15	2	27758	3	2	8784

4×4-bit S-box (Table 10.10) is applied 16 times in parallel to the intermediate cipher state; a linear transformation, called pLayer, consisting of a fixed bit permutation. Only the XOR layer with round subkeys is an involution. Thus, the decryption operation requires the inverse of the S-box and of the pLayer. After the 31-st round, there is an output transformation consisting of an XOR with the last round subkey. One full round of PRESENT is depicted in Figure 10.4.

Let $x = (x_3, x_2, x_1, x_0)$ denote an input nibble to an S-box, and $y = (y_3, y_2, y_1, y_0)$ denote its output nibble such that $\mathsf{S}[x] = y$. The algebraic expressions of the y_i's in terms of the x_j's and vice versa are depicted in Figure 10.5, where addition and multiplication are performed over $\mathsf{GF}(2)$.

Notice how y_0 and x_0 are much simpler functions than the other bits. In particular, y_0 has non-linear degree two (quadratic) and depends linearly on x_0 and x_3. Similarly, x_0 has non-linear degree two and depends linearly on y_0 and y_2.

Table 10.8: Attacking 8-round LBlock with 7 pairs and 32 LSB key bits guessed.

I	n	m_0	T	n_L	n_c
1	10128	32815	23065	8680	8680
2	1448	32815	36740	300	8980
3	1148	32815	42889	228	9208
4	920	32815	48974	157	9365
5	763	32815	58471	111	9476
6	652	32815	55476	47	9523
7	605	32815	51967	20	9543
8	585	32815	47625	25	9568
9	560	32815	36163	141	9709
10	419	32815	27254	126	9835
11	293	32815	16116	145	9980
12	148	32815	4960	142	10122
13	6	32815	8	6	10128

Table 10.9: Attacking 8-round LBlock with 8 pairs and 32 LSB key bits guessed.

I	n	m_0	T	n_L	n_c
1	11472	37945	26201	9832	9832
2	1640	37945	41917	343	10175
3	1297	37945	47974	268	10443
4	1029	37945	55084	186	10629
5	843	37945	65625	129	10758
6	714	37945	63385	57	10815
7	657	37945	57671	22	10837
8	635	37945	50898	21	10858
9	614	37945	40883	161	11019
10	453	37945	30905	144	11163
11	309	37945	19850	160	11323
12	149	37945	5108	145	11468
13	4	37945	6	4	11472

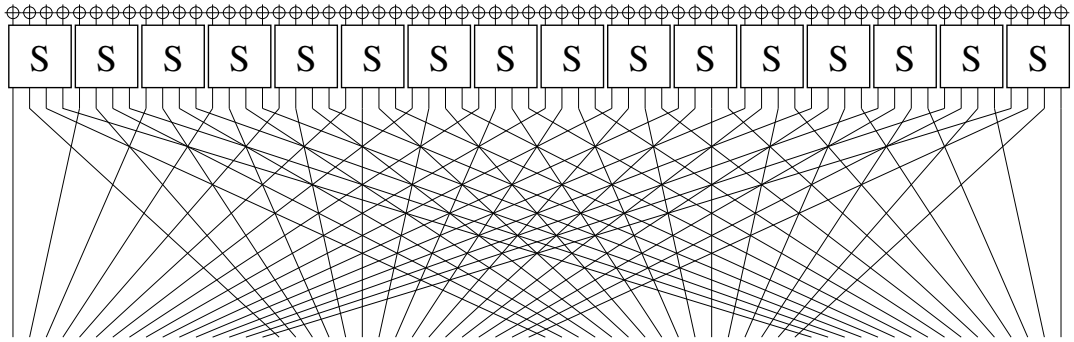


Figure 10.4: One full round of PRESENT.

PRESENT accepts a key of either 80 or 128 bits. Firstly, we focus on the version with 80-bit keys. The user-supplied key is stored in a key register K and is represented as $k_{79}k_{78}\dots k_0$. At round i, the 64-bit round key $K_i = k'_{63}k'_{62}\dots k'_0$ consists of the 64 leftmost bits of the current contents of the register K. Thus, at round i we have: $K_i = k'_{63}k'_{62}\dots k'_0 = k_{79}k_{78}\dots k_{16}$. After extracting the round key K_i, the key register $K = k_{79}k_{78}\dots k_0$ is updated as follows:

1. $[k_{79}k_{78}\dots k_1k_0] = [k_{18}k_{17}\dots k_{20}k_{19}]$
2. $[k_{79}k_{78}k_{77}k_{76}] = \mathsf{S}[k_{79}k_{78}k_{77}k_{76}]$
3. $[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus r$

where r is the round counter. Thus, the key register is rotated by 61 bit positions to the left, the left-most four bits are given to the PRESENT S-box and the round counter value i is XORed with the bits $k_{19}k_{18}k_{17}k_{16}k_{15}$ of K, where the least significant bit of the round counter is on the right.

For a 128-bit key version of PRESENT, the user-supplied key is stored in a key register K and is represented as $k_{127}k_{126}\dots k_0$. At round i, the 64-bit round key $K_i = k'_{63}k'_{62}\dots k'_0$ consists

Table 10.10: The 4×4-bit S-box of PRESENT and the inverse S-box.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathsf{S}[x]$	12	5	6	11	9	0	10	13	3	14	15	8	4	7	1	2
$\mathsf{S}^{-1}[x]$	5	14	15	8	12	1	2	13	11	4	6	3	0	7	9	10

Figure 10.5: Algebraic expression of output and input bits of the S-box of PRESENT.

$$\mathsf{S}: \begin{cases} y_0 = x_1x_2 + x_0 + x_2 + x_3 \\ y_1 = x_0x_1x_3 + x_0x_2x_3 + x_0x_1x_2 + x_1x_3 + x_2x_3 + x_1 + x_3 \\ y_2 = x_0x_1x_3 + x_0x_2x_3 + x_0x_1 + x_0x_3 + x_1x_3 + x_2 + x_3 + 1 \\ y_3 = x_0x_1x_2 + x_0x_1x_3 + x_0x_2x_3 + x_1x_2 + x_0 + x_1 + x_3 + 1 \end{cases}$$

$$\mathsf{S}^{-1}: \begin{cases} x_0 = y_1y_3 + y_0 + y_2 + 1 \\ x_1 = y_0y_1y_2 + y_0y_1y_3 + y_0y_2y_3 + y_0y_2 + y_1y_3 + y_2y_3 + y_0 + y_1 + y_3 \\ x_2 = y_0y_1y_2 + y_0y_1y_3 + y_0y_2y_3 + y_0y_1 + y_0y_2 + y_1y_2 + y_0y_3 + y_1y_3 + y_3 + 1 \\ x_3 = y_0y_1y_2 + y_0y_2y_3 + y_0y_1 + y_0 + y_1 + y_2 + y_3 \end{cases}$$

of the 64 leftmost bits of the current contents of the register K. Thus, at round i we have: $K_i = k'_{63}k'_{62}\ldots k'_0 = k_{127}k_{126}\ldots k_{64}$. After extracting the round key K_i, the key register $K = k_{127}k_{126}\ldots k_0$ is updated as follows:

1. $[k_{127}k_{126}\ldots k_1k_0] = [k_{66}k_{65}\ldots k_{68}k_{67}]$
2. $[k_{127}k_{126}k_{125}k_{124}] = \mathsf{S}[k_{127}k_{126}k_{125}k_{124}]$
3. $[k_{123}k_{122}k_{121}k_{120}] = \mathsf{S}[k_{123}k_{122}k_{121}k_{120}]$
4. $[k_{66}k_{65}k_{64}k_{63}k_{62}] = [k_{66}k_{65}k_{64}k_{63}k_{62}] \oplus r$

Thus, the key register is rotated by 61 bit positions to the left, the left-most eight bits are yielded to two PRESENT S-boxes, and the round counter value i is XORed with the bits $k_{66}k_{65}k_{64}k_{63}k_{62}$ of K, where the least significant bit of the round counter is on the right.

The designers of PRESENT in [BKL+07] have mentioned that they were unsuccessful in obtaining any satisfactory result in reasonable time using algebraic cryptanalysis (F4 algorithm under MAGMA [Mag]) to break two rounds of a smaller version of the cipher having only seven S-boxes per round compared to the real PRESENT cipher having sixteen S-boxes per round.

It is a straightforward procedure to demonstrate that every 4×4 S-box has at least 21 quadratic equations. The larger the number of equations, the weaker the S-box. In fact, the S-box of PRESENT has exactly 21 equations. Writing the whole 80-bit key variant of PRESENT as a system of quadratic equations for 5 rounds, we obtained 740 variables and 2169 equations. We introduce an attack on both PRESENT with key sizes of 80 and 128 bits. Notice that for both key sizes one pair is not enough to recover the key uniquely and we need at least two pairs.

The summary of our results is presented in Table 10.11. As it is depicted in Table 10.11, the timing results of ElimLin and PolyBoRi are comparable except the time in which PolyBoRi crashed

due to running out of memory. As our experiments revealed, in all cases ElimLin used much less memory compared to F4 under PolyBoRi framework.

Table 10.11: Algebraic attack complexities on reduced-round PRESENT.

N_r	$\#key$	g	Running Time (in hours)	Data	Attack notes
5	80	40	0.043	5 KP	ElimLin
5	80	40	0.067	5 KP	PolyBoRi
5	80	37	0.613	10 KP	ElimLin
5	80	37	0.523	10 KP	PolyBoRi
5	80	36	**3.532**	16 KP	ElimLin
5	80	36	crashed	16 KP	PolyBoRi
5	80	35	**1.845**	16 KP	ElimLin
5	80	35	crashed	16 KP	PolyBoRi
5	128	88	0.050	5 KP	ElimLin
5	128	88	0.069	5 KP	PolyBoRi

Skimming through Table 10.11 reveals that the time it takes to recover 45 bits of the key is less than that of 44 bits. This seems very surprising at first glance, but it can be justified by considering that the running time of ElimLin is highly dependent on the sparsity of the equations. So, our intuition is that as we have picked distinct plaintext and key randomly in each experiment, by pure accident the former system of equations turns out to be sparser than the latter and it is also probable that more linear equations are generated due to some combination of randomly picked plaintexts and keys.

We tried to break 6 rounds of PRESENT by ElimLin and PolyBoRi, but ElimLin did not give us any satisfactory result and PolyBoRi crashed after a while due to running out of memory for 6-round PRESENT.

10.6 A Comparison Between ElimLin and PolyBoRi

Gröbner basis is currently one of the most successful methods for solving polynomial systems of equations. However, it has its own restrictions. The main bottleneck of the Gröbner basis techniques is the memory requirement and therefore, most of the Gröbner basis attacks use relatively small number of samples. It is worthwhile to mention that ElimLin is a subroutine is Gröbner basis computations. But, ElimLin algorithm as a single tool requires a large number of samples to work.

The Gröbner basis solves the system by reductions according to a pre-selected ordering, which can lead to high degree dense polynomials.

ElimLin uses the fact that multiple samples provide an additional information to the solver, and therefore, the key might be found even if we restrict the reduction to degree 2.

Next, we compare the current state of the art implementation of F4 algorithm PolyBoRi and our implementation of ElimLin. In the cases where ElimLin behaves better than PolyBoRi, it does not mean that ElimLin is superior to F4 algorithm. In fact, it just means that there exists a better implementation for ElimLin than for F4 for some particular systems of equations. F4 uses a fixed

ordering for monomials and therefore it does not preserve the sparsity in its intermediate steps. On the other hand, our implementation of ElimLin performs several sparsity preserving techniques by changing the ordering. This drops the total number of monomials and makes it memory efficient.

Table 10.1 and Table 10.11 show that PolyBoRi requires too much memory and crashes for a large number of samples. At the same time, our implementation of ElimLin is slightly slower than PolyBoRi implementation attacking 10 samples of 5-round PRESENT-80 as in Table 10.11. This demonstrates that our implementation of ElimLin can be more effective than PolyBoRi and vice versa, depending on memory requirements of PolyBoRi. However, whenever the system is solvable by our implementation of ElimLin, our experiments revealed that PolyBoRi does not give a significant advantage over ElimLin because the memory requirements are too high.

While PolyBoRi may yield a solution for a few samples, the success of ElimLin is determined by the number of samples provided to the algorithm. The evaluation of the number of sufficient samples in ElimLin is still an open problem.

We often see that preserving the degree by simple linear algebra techniques can outperform the more sophisticated Gröbner basis algorithms, mainly due to the structural properties of the system of equations of a cryptographic primitive (such as sparsity). ElimLin takes advantage of such structural properties and uncovers some hidden linear equations using multiple samples. According to our experiments, PolyBoRi does not seem to be able to take advantage of these structural properties as would be expected. This results in higher memory requirements than would be necessary and ultimately failure for large systems, even though it is clearly possible for the algorithm to find the solution in reasonable time. Finally, we need more efficient implementations and data structures for both ElimLin and Gröbner basis algorithms.

Chapter 11

SAT Solving Techniques with Applications to Cryptanalysis of KATAN

11.1 Introduction

The area of SAT Solving has seen tremendous progress over the last years. Many problems (e.g. in hardware and software verification) and in our application in cryptanalysis that seemed to be completely out of reach a decade ago can now be handled routinely. Beside new algorithms and better heuristics, refined implementation techniques turned out to be vital for this success. New SAT solvers can now solve large systems in reasonable time. Since 2002, almost each year a SAT Race competition [SAT] has been established. In 2007 and 2010 respectively, MiniSat [ES05] and CryptoMiniSat [NS09] won the Gold prizes. We used these two SAT solvers in our analysis, but since the timings of MiniSat were faster, we do not report CryptoMinisat results.

An instance of an MQ problem can be simply changed to an instance of a SAT problem. In fact, both these problems are NP hard for a random system. As mentioned earlier, many cryptographic protocols can be formulated as a large system of multivariate equations in which their security is based on the security of the MQ problem. One example is QUAD, a provably secure stream cipher [BGP06]. Its security is directly determined by the complexity of solving a large multivariate polynomial system of equations.

In this chapter, we are going to represent the block cipher KATAN [DCDK09] as an instance of an MQ problem and then change it to an instance of a SAT problem. We propose a new pre-processing on its system of equations before feeding it to MiniSat. This pre-processing stage makes the SAT solver run faster and allows us to break a higher number of rounds of KATAN.

The results in this chapter have been published in INDOCRYPT 2010 [BCN+10].

11.2 Algebraic Attacks Using SAT Solvers

To solve polynomial systems of multivariate equations by SAT solvers, the attacker initially converts the system from Algebraic Normal Form (ANF) to Conjunctive Normal Form (CNF). There is an efficient conversion method due to Bard, Courtois and Jefferson (BCJ) [BCJ07]. The con-

catenation of these CNFs gives a file with extension .cnf on which we can apply any SAT solver.

11.3 Algebraic Cryptanalysis of KATAN Family of Block Ciphers

KATAN is a family of lightweight, hardware-oriented block ciphers consisting of three variants with 32, 48 and 64-bit blocks. For all KATAN ciphers, the key size is 80 bits ($n = 80$) and they all iterate 254 rounds [DCDK09]. The block size is used as a suffix to designate each cipher member, as KATAN32, KATAN48 and KATAN64. The design of these ciphers was inspired by the stream cipher Trivium [DCP08]. The structure of KATAN32 consists of two LFSR's, called L_1 and L_2, loaded with the plaintext and then transformed by two non-linear Boolean functions, f_a and f_b as follows (Table 11.1 lists the bit sizes and the indices x_i and y_j of L_1 and L_2):

$$f_a(L_1) = L_1[x_1] + L_1[x_2] + L_1[x_3] \cdot L_1[x_4] + L_1[x_5] \cdot \mathrm{IR} + k_a$$
$$f_b(L_2) = L_2[y_1] + L_2[y_2] + L_2[y_3] \cdot L_2[y_4] + L_2[y_5] \cdot L_2[y_6] + k_b$$

where IR is the output of an LFSR, i.e., $L_1[x_5]$ is used whenever IR = 1. The values of IR for each round is specified in [DCDK09]. For the i-th round, $k_a = k_{2i}$ and $k_b = k_{2i+1}$ that is, only two key bits are used per round. The output of each of these functions is loaded to the least significant bits (LSB) of the other LFSR, after they are left-shifted. This operation is performed in an invertible manner (see Figure 11.1).

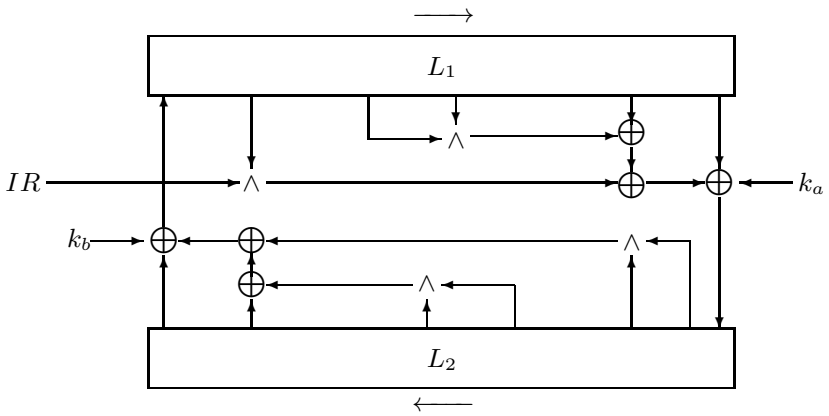


Figure 11.1: The outline of a round of the KATAN/KTANTAN ciphers.

For KATAN48, f_a and f_b are each applied twice per round, so that the LFSR's are clocked twice; however, the same pair of key bits are reused. For KATAN64, each Boolean function is applied three times per round, again with the same pair of key bits reused three times.

The selection of bits x_i and y_i in f_a and f_b are listed in Table 11.1. Plaintext and ciphertext bits are numbered in **right-to-left order** starting from 0. Thus, for instance, a plaintext block for KATAN32 will be numbered as $p = (p_{31}, \ldots, p_0)$. The key schedule algorithm of all KATAN ciphers is a linear mapping that expands an 80-bit key K to 508 subkey bits according to

$$k_i = \begin{cases} K_i, & \text{for } 0 \leq i \leq 79 \\ k_{i-80} + k_{i-61} + k_{i-50} + k_{i-13} & \text{otherwise} \end{cases}$$

Table 11.1: Parameters for the f_a and f_b functions.

Cipher	$\|L_1\|$	$\|L_2\|$	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
KATAN32	13	19	12	7	8	5	3	18	7	12	10	8	3
KATAN48	19	29	18	12	15	7	6	28	19	21	13	15	6
KATAN64	25	39	24	15	20	11	9	38	25	33	21	14	9

Thus, the subkey of the i-th round is $k_a||k_b = k_{2i}||k_{2i+1}$.

After r rounds, at most $2r$ key bits are mixed with the internal state, as two key bits are XORed per round. Thus, at least 40 rounds are needed before a complete key diffusion for any KATAN cipher is achieved. Further details about these ciphers can be found in [DCDK09]. For analysis purposes, the numbering in the user key in our attacks is $K = (K_{79}, \ldots, K_0)$.

11.3.1 Straightforward Algebraic Attack on KATAN Using SAT Solvers

One instance of KATAN32 can be represented as 8 620 very sparse quadratic equations with 8 668 variables, KATAN48 as 24 908 equations and 24 940 variables and KATAN64 as 49 324 equations and 49 340 variables. As can be observed, the system is underdefined. That is because the key size is larger than the block size for all versions. To have a defined or an overdefined system, we need multiple samples.

The summary of our results is in Table 11.2. We used the "guess and determine" algebraic attack initially proposed in [CB07]. This implies that we fix g bits of the key and then we show that recovering the other $80-g$ bits is faster than exhaustive search (the same approach as of Chapter 10, Sec. 10.5). This is represented in the column titled "g" in Table 11.2. In fact, we fix g LSB of the key, as heuristically we obtained better results than fixing the g MSB of the key. We used the graph partitioning method by Wong and Bard [WB10] to derive the best state variables to fix, but it did not bring about anything better than using the heuristic of fixing the least g significant bits of the key. Note, if we fix g bits, the algebraic attack is solving a system of equations to recover the $80 - g$ remaining bits. The time to recover this $80 - g$ bits is represented as *"Running Time"* in Table 11.2.

We represent the time complexity of the SAT Solver (MiniSat) in seconds using a 3 Ghz CPU. Note that our algebraic attacks are in a chosen-plaintext scenario, except in some rare cases as noted. We noticed that chosen-plaintext attack is much stronger against KATAN family than known-plaintext (KP) attack. In our attacks, we used the following structure for the chosen plaintexts for KATAN32: $p_{i+1} = ((p_i \gg 19) + 1) \ll 19$ and $p_{i+1} = ((p_i \gg 29) + 1) \ll 29$ for KATAN48 and $p_{i+1} = ((p_i \gg 39) + 1) \ll 39$ for KATAN64 for $i \geq 1$, where p_i is the i-th plaintext we pick and p_1 is random. Note that bits $19, 29$ and 39 are exactly the bit 0 of L_1 register for KATAN32, KATAN48 and KATAN64 respectively. This choice of the bits makes the SAT solver run faster. Moreover, we believe it is fair to assume that each round encryption of KATAN takes at least 5 CPU cycles. This yields a comparison between the complexity of our attacks and exhaustive key search.

Deploying the straightforward method for converting ANF to CNF and then feeding it to a SAT solver, we could break up to 75 rounds of KATAN32 and 64 rounds of KATAN48 and 60 rounds of KATAN64. But, we can do better by performing a pre-processing on the system of

Table 11.2: Attack complexities on KATAN family of block ciphers (memory complexity is negligible).

Cipher	N_r	Running Time	Data	g	Attack	Source
KATAN32	40	11 sec	3 KP	0	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	50	11 sec	3 KP	0	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	60	18 sec	3 KP	0	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	65	1.81 min	3 KP	0	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	66	8.85 min	3 KP	0	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	67	26 sec	3 KP	30	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	68	2.55 min	3 KP	30	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	69	47.76 min	3 KP	35	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	70	1.64 min	10 CP	35	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	71	3.58 min	10 CP	35	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	75	12.50 h	3 CP	35	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	76	1.59 min	20 CP	45	MiniSat2.2, BCJ conv./Pre-Pro.	Sect. 11.3.2
	76	4.1 min	20 CP	43	MiniSat2.2, BCJ conv./Pre-Pro.	Sect. 11.3.2
	76	3.08 min	20 CP	41	MiniSat2.2, BCJ conv./Pre-Pro.	Sect. 11.3.2
	77	18 sec	20 CP	45	MiniSat2.2, BCJ conv./Pre-Pro.	Sect. 11.3.2
	78	5.80 min	20 CP	45	MiniSat2.2, BCJ conv./Pre-Pro.	Sect. 11.3.2
	79	14.72 min	20 CP	45	MiniSat2.2, BCJ conv./Pre-Pro.	Sect. 11.3.2
KATAN48	40	2 sec	5 CP	40	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	50	7 sec	5 CP	40	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	60	13.18 min	5 CP	40	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	61	7.12 min	5 CP	45	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	62	11.86 min	10 CP	40	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	63	17.47 min	10 CP	45	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	64	6.42 h	5 CP	40	MiniSat2.2, BCJ conv.	Sect. 11.3.1
KATAN64	40	2 sec	5 CP	40	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	50	12 sec	5 CP	40	MiniSat2.2, BCJ conv.	Sect. 11.3.1
	60	3.17 h	5 CP	40	MiniSat2.2, BCJ conv.	Sect. 11.3.1

Running Time: the time to recover $80 - g$ bits of the key.
KP: known plaintext
CP: chosen plaintext
BCJ: the Bard-Courtois-Jefferson converter
Pre-Pro: the pre-processed system of equations

equations before applying it to a SAT solver. Using this pre-processing (see the next section), we could break 79 rounds of KATAN32. We only tried this method on the KATAN32 equations. Further research would apply this technique to other members of the family.

11.3.2 The Pre-processing SAT-Solver Attack

In this section, we use the equations generated as described earlier and solve them with the SAT solver MiniSat [ES05]. It is simpler to formulate KATAN as a sparse system. But, this may not be the best representation for a SAT solver. One characteristic of our representation is that there are many equations of the form $x = y$, as well as $x = 0$, $y = 1$ and more rarely $x + y = 1$. Also, in a typical example (78 rounds, 45 key bits fixed and 20 CPs of KATAN32) there are 51 321 total equations. Naturally, one would want to take advantage of these special equations during the pre-processing to create a smaller system which has fewer variables and equations.

More precisely, the four heuristics of a CNF problem are (1) the number of variables, (2) the number of clauses, (3) the average number of symbols per clause and (4) the total number of symbols in the system. The pre-processing algorithm that we describe in the next section is designed on the principle of primarily reducing (1) and (2) while causing the minimum possible increase in (3) and (4). To be specific, at each iteration, a substitution will be made and this substitution reduces (1) and (2) by one and the substitution is selected in the style of the "greedy algorithm" using (4) as the criterion.

The following pre-processing algorithm is a refinement of the "massaging" algorithm of [CB07] and so we call it "turbo-massage". Starting with the equations that were generated, we ran the pre-processing algorithm; after that, we converted the polynomials into a CNF problem, according to [CB07] and ran MiniSat on that CNF problem to get a solution. We will explain the pre-processors here and refer the reader to [CB07] or [Bar09] for the process of converting a polynomial system into a CNF problem.

11.3.3 The Turbo-Massage Pre-processing Algorithm

As described before, the equations can be thought of as a series of polynomials, named $f_1(\vec{x}) = 0$, $f_2(\vec{x}) = 0$, ... We define the operation "fully-substitute" as follows: Let $f(\vec{x})$ be a polynomial with some monomial μ. To fully-substitute $f(\vec{x})$ into $g(\vec{x})$ on μ means to

- Write $g(\vec{x})$ in the form $g(\vec{x}) = \mu h_1(\vec{x}) + h_2(\vec{x})$.
- Write $f(\vec{x})$ in the form $f(\vec{x}) = \mu + h_3(\vec{x})$.
- Replace $g(\vec{x})$ with $h_1(\vec{x})h_3(\vec{x}) + h_2(\vec{x})$, which is mathematically equivalent, because in any satisfying solution $\vec{x}$, we would have $\mu = h_3(\vec{x})$.
- By clever use of data structures, this can be made highly efficient.

Observe that for the four common forms: $x = y$, as well as $x = 0$, $y = 1$ and more rarely $x+y = 1$, the "fully-substitute" definition does what one would do if solving a system of equations with a pencil and paper. For more higher weight $f(\vec{x})$, understanding what it does to $g(\vec{x})$ is more complex.

We must also use a non-standard definition of the weight of a polynomial $f(\vec{x})$. We define it to be the number of monomials in $f(\vec{x})$, but excluding the constant $+1$ from the tabulation. The reason for this is that if $f(\vec{x})$ has weight w according to this modified definition, then 2^{w-1} conjunctive normal form clauses of length w will be required to represent the polynomial, assuming

all the monomials are already defined. The total number of symbols is then $w2^{w-1}$, and so minimizing w is crucial in keeping the CNF problem small and thus solvable. This procedure is represented in Algorithm 11.1.

Algorithm 11.1 The Turbo-Massage Pre-processing algorithm.

Input: A system of polynomial equations over GF(2), and a weight-limit w_{max}.
Output: A system of polynomial equations over GF(2) with maximum weight w_{max}.
Mark all polynomials "unused."
while the set of unused polynomials is not empty **do**
  Locate the lowest weight unused polynomial $f(\vec{x})$.
  if $f(\vec{x})$ exceeds w_{max} **then**
    Terminate.
  end if
  Mark $f(\vec{x})$ as "used".
  if $f(\vec{x})$ has weight 1 **then**
    select μ to be the only monomial in $f(\vec{x})$.
  end if
  if $f(\vec{x})$ has weight 3 or higher **then**
    select μ to be the monomial which appears least frequently in the entire system of equations.
  end if
  if $f(\vec{x})$ has weight 2 **then**
    select μ to be the monomial which appears most frequently in the entire system of equations.
  end if
  for any polynomial $g(\vec{x})$ containing the monomial μ **do**
    "fully-substitute" $f(\vec{x})$ into $g(\vec{x})$ on μ.
  end for
end while

The "turbo-massaging" algorithm will always terminate, because eventually, every polynomial has been marked used. In practice, it will terminate early, where all unused polynomials are of weight w_{max} or higher. If there are n "used" polynomials, then there will be n monomials which appear nowhere in the entire system, except in exactly one polynomial. This is of course the monomial μ which was chosen when that polynomial was getting used. In our system of equations, it was almost always the case (by an overwhelming margin) that μ was degree one. And so, each used polynomial effectively amputates one variable from the polynomial system of equations.

The special case of weight 2 deserves an explanation. When f has weight 1, there is no decision to be made, but it is noteworthy that the weight of g will decrease. When f has weight 2, then the weight of g will not change during the "fully-substitute" operation, except in some odd cases like substituting $x = y$ into $zx + zy + w + x + y = 0$, where the weight goes from 5 to 1 instantly. As the weight is not likely to change and we are eliminating a monomial, it makes sense to eliminate a common monomial. When the weight of f is 3 or more, then the weight of g will increase. If we choose μ to be very popular, appearing k times, then the total weight of the system will increase by $k(w-2)$. Thus, it makes sense to keep the weight growth bounded and choose μ to be rare. This heuristic was found after an enormous number of iterations of "trial-and-error".

For example, in the 78-round, 20 CP, 45-bit-key case of KATAN32, the weight went from 110 726 to 101, 516 after 47 032 polynomials got used. Furthermore, the "fully substitute" function was called 330 587 times. There were 50 033 equations at this point, down from 51 321, representing 1 288 equations that became $0 = 0$. In other words, the original system was not full rank. The average weight of a polynomial, using the modified definition of weight, was roughly 2.02898. The system had 53 993 distinct monomials, plus 2 005 variables which appeared only in degree 1 monomials and ended with a CNF problem of 55 398 variables and 156 010 clauses. The conversion process, which must be run only once and not 2^{45} times, takes between 20 and 29 minutes in all the cases explored here.

It should be noted that in other polynomial systems, it might be the case that μ is often quadratic or higher degree. It remains open if one should force μ to be linear when possible. This is a question that the authors hope to investigate shortly. As it comes to pass, $w_{max} = 2$ turned out to be slightly better than $w_{max} = 3$ for this problem, but in other cases up to $w_{max} = 5$ has been used.

A minor note for algebraic geometers familiar with the concept of a Macaulay matrix [Mac16] in the Lazard [Laz83] family of algorithms (including F4 [Fau99], XL and their variants [CSPK00, Cou08]) is that this algorithm is like a Gaussian Elimination on that matrix, but stopping early. The pivoting strategy used is reducible to the Markowitz pivoting algorithm [Mar57]. However, the "fully-substitute" is not the same in this case, as adding $x = y$ to $zx+zy+w+x+y = 0$ would result in $zx + zy + w = 0$. On the other hand, fully-substituting $x = y$ into $zx + zy + w + x + y = 0$ would result in $zx + zx + w + x + x = 0$ which turns into $w = 0$. As can be seen, full-substitution is distinct from adding and is very similar to what a mathematician would do when solving a system of polynomial equations with a pencil and paper.

11.3.4 Better Cryptanalysis Results on KATAN32

The first result was 76 rounds, 20 CP and 45 (fixed) key bits of KATAN32, broken faster than by brute force. To extend this result, we explored using fewer key bits and more rounds. First, we conducted the above process for 20 CPs and for 76, 77, 78, 79 and 80 rounds. Every case was run 50 times.

Because we fixed 45 bits of the key, and so assuming one nano-second per round for a brute force attacker using a 3 Ghz CPU, our attack against r rounds is faster than brute force, if and only if it runs in t seconds with $2^{45}t < r2^{80}10^{-9}$ or more plainly $t < r2^{35}10^{-9} \approx r(34.3597\cdots)$. We also ran trials with 43 bits of the key fixed for 76 rounds and there the threshold would be 4 times greater or $137.439r$ seconds and for 41 bits of the key $549.755r$ seconds.

The running times are given in Table 11.3. Observe the enormous variance in each trial. In some cases, the fastest run is 1000× faster than the slowest. This is very typical in SAT-solver-based cryptanalysis. We excluded the three fastest and slowest trials, and took the mean and standard deviation of the remaining 44 trials.

The running time of 2^{45} executions, all added together, is the sum of 2^{45} samples from independent random variables. Therefore, the central limit theorem applies and regardless of the actual distribution of the running times, if the mean is m_1 and the standard deviation (stdev) is σ_1, the sum of 2^{45} of them will be normally distributed and have a mean of $2^{45}m_1$ and a standard deviation of $2^{22.5}\sigma_1$. As σ/m is an important instrument in gauging the reliability of a normal sample, it is interesting to note here that σ/m (for the sum of 2^{45} execution times) would be $2^{-22.5}(\sigma_1/m_1)$

which is phenomenally tiny. Thus, the running time of the real-world attacker would be essentially constant.

Notice, that we claim that the 2^{45} running times are independent, but we do not claim that they are identically distributed. On the other hand, one could conceive a cipher where one key bit was ignored by the cipher, in which case the running times for two keys, which differ only in that bit would be highly dependent. These cases are of pedagogical interest only because no cipher designer would ever do that.

As can be seen in Table 11.3, we are between 80.75 and 2.39 times faster than brute force search for up to and including 79 rounds. In the case of 80 rounds, out of 50 trials, 29 of them timed-out after 1 hour. As this is a majority, it is not possible that the mean is less than the required 2 748.77 seconds, and so we are not faster than brute-force for 80 rounds. For 43 key bits and 41 key bits, the attack becomes vastly more efficient. But, we cannot test 39 key bits, as the time-out value would have to be set to 167,125 seconds or roughly 46 hours, for each of 50 processes.

In addition to MiniSat, we ran all 50 instances with CryptoMiniSat [NS09], a SAT-Solver constructed specifically for cryptography by Mate Soos. However, it was consistently slower than MiniSat. We suspect that this is the case because CryptoMiniSat was intended to minimize the impact of long-XORs, which are normally very damaging to the running time of SAT-solver methods; however, we have no long-XORs in our equations, in fact, no sum was longer than 5 symbols after pre-processing, excluding the constant monomial.

11.3.5 The Gibrat Hypothesis

In [CB07], [CM03] as well as [Bar09], Bard hypothesized that the true distribution of the running times of a CNF-problem in a polynomial-system-based SAT problem follows the Gibrat distribution. That is to say, that the logarithm of the running time is normal. The running times here were such that their standard deviations exceeded the mean. If the distribution of the running time was normal, having $\sigma > \mu$ would imply a very significant fraction of the running times would be negative. Therefore, it is not possible that the running time is normally distributed. On the other hand, we also tabulated the mean and standard deviation of the logarithm.

The ratio of the mean and standard deviation of the logarithm of running times is much more reasonable. The kurtosis is the typical measurement of the "normalness" of a distribution and the kurtosis of the logarithms of the running times are far closer to 1 (and are in fact within ± 1) than the kurtosis of the running times themselves (which had kurtoses over 9). So the hypothesis that the running times are Gibrat, from [CB07], seems well-justified for these examples.

11.3.6 A Strange Phenomenon

We were perplexed to discover that solving 77 rounds was far easier than solving 76 rounds or 78 rounds of KATAN32. Therefore, we ran the experiments again, with both sets of results listed in Table 11.3 as the first batch and the second batch. As can be seen, in both cases, 77 rounds is much easier than 76 or 78 and with a very large margin. Moreover, this remained true as well in our experiments with CryptoMiniSat. As random variables, the i-th iteration of the 76 round attack and the i-th iteration of the 77 round attack had absolute correlation of $0.060419\ldots$ and likewise between 77 and 78 it was $-0.09699\cdots$. These extremely low correlations make it safe to

hypothesize that the running times are independent and this removes the possibility that the effect is an artifact of some methodology error. Note that the formula for correlation that we used is

$$\mathrm{Cor}(X, Y) = \frac{\mathrm{E}[(X - \mu_x)(Y - \mu_y)]}{\sigma_x \sigma_y}$$

as is standard. Moreover, we observed the same behaviour when dealing with the size of the vertex separator in the variable-sharing graph representation of the polynomial system of equations of KATAN32 using the strategy described in [WB10]. For KATAN32, the size of the vertex separator is not increasing with the number of rounds, and as a matter of fact, it fluctuates. We offer no explanation to the cause of the weakness of the 77-round version of KATAN32.

Table 11.3: Running time and some statistical results for different number of rounds of the pre-processed equations for KATAN32 (The running times are in second).

N_r	76	77	77	78	79	80	76	76
g	45	45 (1st batch)	45 (2nd batch)	45	45	45	43	41
1	2.89	1.00	2.43	11.04	17.05	59.62	1.50	1.75
2	3.15	2.16	3.69	11.54	24.97	64.61	5.48	1.91
3	3.39	2.25	4.01	14.51	26.86	100.28	15.75	3.36
4	3.39	3.39	4.12	15.83	28.82	135.34	25.88	3.77
5	4.61	3.93	4.40	19.17	54.27	157.10	34.81	5.17
6	6.73	4.16	4.44	24.99	57.02	166.41	39.92	5.65
7	8.29	4.22	4.65	51.46	60.72	230.60	39.97	8.64
8	8.46	4.58	4.72	63.04	64.08	277.04	45.06	11.35
9	11.54	4.81	5.07	86.06	70.34	353.45	50.19	21.71
10	13.15	4.84	6.41	89.89	89.17	354.07	50.79	35.31
11	17.19	4.96	6.81	109.21	109.86	402.56	52.09	41.71
12	17.62	5.44	10.08	115.86	130.28	423.76	60.94	53.7
13	23.64	5.62	14.54	141.19	137.77	433.73	75.35	55.77
14	26.60	5.74	15.03	148.91	145.05	463.78	102.91	61.6
15	27.69	5.83	18.16	161.49	210.29	516.65	116.01	78.29
16	37.32	6.80	18.51	163.23	217.28	687.88	121.89	84.18
17	38.04	7.64	19.51	206.66	269.08	1163.48	123.25	87.51
18	39.67	8.38	21.31	218.43	326.69	1591.56	123.36	104.76
19	48.68	9.54	21.35	230.86	402.61	2180.93	124.39	108.29
20	50.63	10.08	21.57	236.17	408.39	3261.20	131.54	128.62
21	56.51	11.32	22.06	241.45	537.16	3274.25	132.67	138.37
22	62.53	13.81	22.41	248.64	547.32	29 timeouts	134.03	166.93
23	66.03	15.72	22.63	256.66	718.58		207.34	170.14
24	81.25	16.69	27.15	293.66	780.44		208.48	182.83
25	88.88	17.47	28.45	319.31	873.25		233.40	183.9
26	101.43	17.86	32.39	377.06	893.29		258.52	185.41
27	115.13	19.19	45.27	455.50	949.06		300.38	200.08
28	127.09	19.63	49.92	504.97	1007.55		326.94	223.6
29	176.33	22.76	54.80	593.65	1223.91		374.62	246
30	200.26	24.29	54.82	822.36	1244.11		387.17	248.05
31	224.75	29.68	73.71	854.80	1388.40		444.42	254.58
32	243.36	30.09	82.72	880.31	1436.00		449.31	256.05
33	258.53	33.27	85.42	1111.59	1632.59		542.73	263.13
34	278.53	34.02	85.56	1118.54	1838.31		829.13	275.75
35	294.99	35.62	97.22	1197.05	1864.98		905.35	304.75
36	353.49	35.94	97.76	1388.38	1875.87		954.94	305.1
37	407.02	43.33	103.34	1449.29	2031.08		1217.79	305.18
38	423.38	43.65	111.18	1514.89	2038.93		1367.94	328.86
39	475.98	48.18	118.48	1517.73	2167.55		1390.52	352.89
40	506.67	48.22	119.15	1533.10	2262.50		1618.79	356.23
41	687.95	49.96	184.91	1538.97	2369.57		2234.32	403.7
42	842.95	73.62	222.26	1689.96	2413.38		2455.77	407.63
43	942.88	106.69	226.48	1894.40	2495.42		2668.97	418.7
44	2387.95	133.21	335.07	2031.93	2641.90		3246.26	427.04
45	2400.12	186.39	456.45	2375.14	2960.11		3326.73	429.21
46	3722.62	201.89	662.92	2682.71	3460.90		3530.63	555.35
47	4471.28	302.66	815.38	2837.97	4023.81		7157.16	577.3
48	> 6000	344.63	976.94	3731.61	4129.64		9378.05	6248.59
49	> 6000	433.70	2378.61	> 6000	4212.65		$> 10,000$	6763.91
50	> 6000	524.56	> 6000	> 6000	> 6000		$> 10,000$	9655.8
Threshold-time	2611.34	2645.70	2645.70	2680.06	2714.42	2748.78	10445.35	41781.40
# faster	47	50	49	48	49	21	48	50
Median	95.16	17.67	30.42	348.19	883.27	n/a	245.96	184.66
Mean of all but 6	463.21	38.98	100.88	768.47	1146.77	n/a	868.70	205.97
Stdev of all but 6	957.09	60.29	168.93	786.56	1054.09	n/a	1381.91	154.29
Kurtosis of all but 6	9.51	9.19	9.30	0.17	-0.12	n/a	9.33	-0.46
Times faster than brute force	5.64	67.87	26.23	3.49	2.37	n/a	12.02	202.85
Mean of log	4.648	2.914	3.650	5.938	6.369	n/a	5.706	4.810
Stdev of log	1.820	1.185	1.421	1.380	1.396	n/a	1.513	1.319
Kurtosis of log	-0.582	-0.470	-0.620	-0.514	-0.985	n/a	-0.985	0.867

Chapter 12

Algebraic Cryptanalysis of ARMADILLO1

12.1 Introduction

ARMADILLO cipher is a hardware oriented multi-purpose cryptographic primitive designed by ORIDAO [Ori] and presented at CHES'10 [BDN+11]. It was built for RFID applications. It can be used as a PRF/MAC, for example for challenge-response protocols as a MAC and also as a hash function for digital signatures or a PRNG for constructing a stream cipher. It has two versions, named ARMADILLO subsequently denoted by ARMADILLO1 and ARMADILLO2.

Due to the attack described in this chapter against ARMADILLO1 and also to shrink the design, multiple intermediate versions of ARMADILLO were designed [SSV11]. Finally, ARMADILLO2 was adopted.

We introduce a generalized version ARMADILLOgen which covers all distinct versions of ARMADILLO and we explain when the key recovery attack is possible.

The attack against ARMADILLO1 is polynomial and has complexity $O(c^2 \log c)$, where c is the size of the key. In fact, it can be performed "by hand", as the actual key recovery algorithm is very simple. There exist multiple other polynomial time key recovery and forgery attacks on distinct versions of ARMADILLO. The interested readers can refer to [SSV11] for more details.

The results in this chapter were published in CHES 2010 [BDN+11] and CARDIS 2011 [SSV11].

In [ABNP+11], the authors found an attack against ARMADILLO2 based on parallel matching. The key recovery attack against FIL-MAC application of ARMADILLO2-A and ARMADILLO2-E using single challenge-response pair is 2^7 and 2^{18} times faster than exhaustive search respectively. Finally, ARMADILLO2 was practically broken recently in [NPP12].

12.2 Description of ARMADILLO

ARMADILLO relies on data-dependent bit transpositions. Given a bitstring x with bit ordering $x = (x_\ell \| \cdots \| x_1)$, fixed permutations σ_0 and σ_1 over the set $\{1, 2, \ldots, \ell\}$, a bit string s, a bit $b \in \{0, 1\}$ and a permutation σ, define $x_{\sigma_s} = x$ when s has length zero and $x_{\sigma_{s\|b}} = x_{\sigma_s \circ \sigma_b}$, where

x_σ is the bit string x transposed by σ, that is,

$$x_\sigma = (x_{\sigma^{-1}(\ell)} \| \cdots \| x_{\sigma^{-1}(1)})$$

The function $(s, x) \mapsto x_{\sigma_s}$ is a data-dependent transposition of x. The function $s \mapsto \sigma_s$ can be seen as a particular case of the general semi-group homomorphism from $\{0,1\}^*$ to a group G.

Notations. Throughout this chapter, $\|$ denotes the concatenation of bitstrings, $\oplus$ denotes the bitwise XOR operation, $\overline{x}$ denotes the bitwise complement of a bitstring x; we assume the little-endian numbering of bits, such as $x = (x_\ell \| \cdots \| x_1)$.

In this section, we give the description of two variants ARMADILLO1 and ARMADILLO2. Then, we introduce a common generalized version ARMADILLOgen and show how it relates to all versions.

12.2.1 ARMADILLO1

ARMADILLO1 maps an initial value C and a message block U to two values (see Figure 12.1) .

$$(V_C, V_T) = \mathsf{ARMADILLO1}(C, U)$$

ARMADILLO1 works based on a register Xinter. By definition, C and V_C are of c bits, V_T as well as each block U_i are of m bits, Xinter is of $k = c + m$ bits. ARMADILLO1 is defined by integer parameters c, m and two fixed permutations σ_0 and σ_1 over the set $\{1, 2, \ldots, 2k\}$. Concretely, we consider $m \geq 40$ and $k = c + m$. To initialize ARMADILLO1, Xinter is set to $C \| 0^m$, where 0^m is a null padding block and C is an initial value. ARMADILLO1 works as follows (Figure 12.1).

1: in the i-th step, replace the rightmost m-bit block of Xinter by the block U_i;
2: set a $\ell = 2k$ bits register $x = \overline{\mathsf{Xinter}} \| \mathsf{Xinter}$;
3: x undergoes a sequence of bit permutations which we denote by P. The output of this sequence of bit permutations is truncated to the rightmost k bits, denoted S, by

$$S = \mathsf{tail}_k((\overline{\mathsf{Xinter}} \| \mathsf{Xinter})_{\sigma_{\mathsf{Xinter}}})$$

4: set Xinter to the value of $S \oplus \mathsf{Xinter}$.
5: after processing the last block U_n, take $(V_C \| V_T) = \mathsf{Xinter}$ as the output.

12.2.2 ARMADILLO2

For completeness, we now provide the description of ARMADILLO2 [BDN+11] here. ARMADILLO2 is mostly based on ARMADILLO1b (to be defined later) with an additional pre-processing mechanism. We note that the pre-processing step outputs a sequence of bits that defines the data dependent permutation and ensures that the data dependent permutation σ_{Xinter} cannot be easily controlled by the attacker (see Figure 12.2).

1: in the i-th step, replace the rightmost m-bit block of Xinter by the block U_i;
2: set a $\ell = k$ bits register $x = \mathsf{Xinter}$;

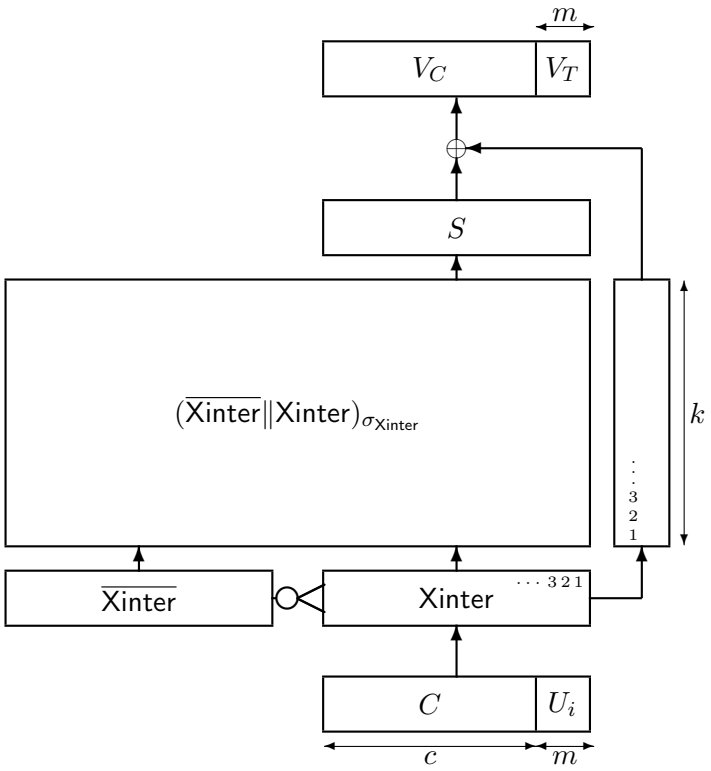


Figure 12.1: Scheme of ARMADILLO1.

3: x undergoes a sequence of bit permutations, σ_0 and σ_1 and a constant γ addition, which we denote by P. In fact, P maps a bitstring of m bits and a vector x of k bits into another vector of k bits as $P(s\|b, x) = P(s, x_{\sigma_b} \oplus \gamma)$, where $b \in \{0, 1\}$ and x_{σ_b} is a permutation of bits of x (transposition). The output of this sequence of k bit permutations and constant addition is denoted $Y = P(U_i, x)$. We call this step pre-processing, as it is used to define the permutation for the consequent step.

4: x undergoes a sequence of bit permutations and constant addition P defined by Y. The output of this sequence of k bit permutations and constant addition is denoted $S = P(Y, x)$.

5: set Xinter to the value of $S \oplus$ Xinter.

6: after processing the last block U_n, take $(V_C\|V_T)$ as the output.

12.3 General ARMADILLOgen Algorithm

In [SSV11], the authors defined various intermediate versions of ARMADILLO. In fact, these distinct versions revealed the design rationale behind the structure of ARMADILLO2. All these versions are based on a data-dependent permutation P. They all can be covered under ARMADILLOgen as a parametrized version of distinct variants and by setting the corresponding parameters, we obtain ARMADILLO1, ARMADILLO1b, ARMADILLO1c, ARMADILLO1d and ARMADILLO2.

ARMADILLOgen is defined as

$$\mathsf{ARMADILLOgen}(X) = T_4(P(T_1(X), T_2(X)), X)$$

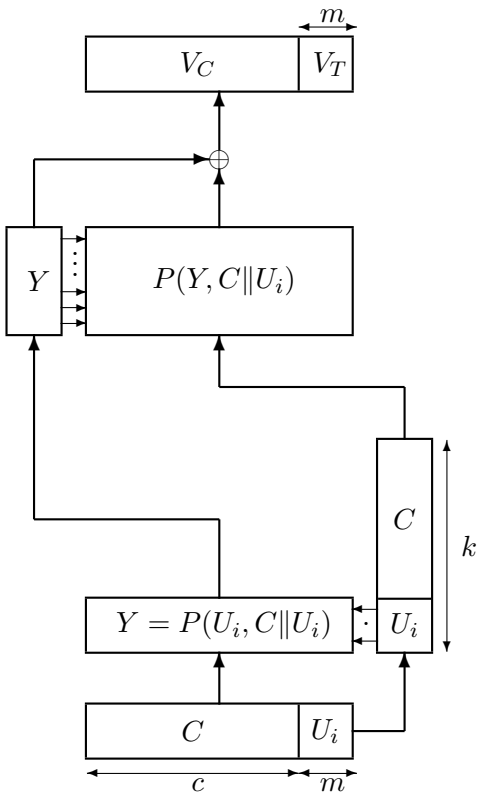


Figure 12.2: Scheme of ARMADILLO2.

where

$$P(s\|b, Y) = P(s, T_3(b, Y))$$
$$P(\lambda, Y) = Y$$

λ denotes the empty string, T_1, T_2 and T_4 are some linear functions and T_3 in its most general form is

$$T_3(b, Y) = L(Y)_{\sigma_b} \oplus \gamma$$

where L is linear and γ is a constant.

Then, ARMADILLO1 is defined as ARMADILLOgen for

$$T_1(X) = X$$
$$T_2(X) = \overline{X}\|X$$
$$T_3(b, X) = X_{\sigma_b}$$
$$T_4(X, Y) = \mathsf{tail}_k(X) \oplus Y$$

ARMADILLO2 is defined as ARMADILLOgen for

$$T_1(X) = P(\mathsf{tail}_m(X), X)$$
$$T_2(X) = X$$
$$T_3(b, X) = X_{\sigma_b} \oplus \gamma$$
$$T_4(X, Y) = X \oplus Y$$

ARMADILLO1c is defined as ARMADILLOgen for

$$T_1(X) = X$$
$$T_3(b, X) = L(X)_{\sigma_b} \oplus \gamma$$

for a linear transformation L with arbitrary linear T_2 and T_4.

ARMADILLO1d is defined as ARMADILLOgen for

$$T_1(X) = X_\pi$$
$$T_3(b, X) = X_{\sigma_b} \oplus \gamma$$

for a fixed permutation π and with arbitrary linear T_2 and T_4.

12.3.1 ARMADILLO1b: Shrinking the Xinter Register

ARMADILLO1b is a compact version of ARMADILLO1 which prevents the preservation of Hamming weight by adding a constant. However, it does not prevent the attack against ARMADILLO1. According to [BDN+11], the ARMADILLO1 design prevents a distinguishing attack based on a constant Hamming weight by having a double sized internal register and a final truncation, assuming the output of the P transposition looks pseudo-random. We see later in this paper (see section 12.4) that this proof does not hold in standard attack model and ARMADILLO1 can be broken in polynomial time. First, we define ARMADILLO1b and then demonstrate an attack against this version and explain how the same attack can be used against ARMADILLO1.

ARMADILLO1b is defined as ARMADILLOgen for

$$T_1(X) = X$$
$$T_2(X) = X$$
$$T_3(b, X) = X_{\sigma_b} \oplus \gamma$$
$$T_4(X, Y) = X \oplus Y$$

In the design of ARMADILLO1b the state size is reduced to k bits to save more gates. So, there is only the register Xinter and not its complement and there is no truncation. To avoid Hamming weight preservation, after each permutation, there is an XOR of the current state with a constant γ (see Figure 12.3).

12.4 Key Recovery Attack on ARMADILLO1 and ARMADILLO1b

In this section, we describe an attack against two versions of ARMADILLO. We first explain the attack on ARMADILLO1b and then by setting $\gamma = 0$ and extending the initial state to $(\overline{\mathsf{Xinter}}\|\mathsf{Xinter}) = (\overline{C}\|\overline{U}\|C\|U)$, the same attack can be directly used against ARMADILLO1.

As ARMADILLO has more than one applications, we briefly explain how it is deployed in the challenge-response application. We refer the reader to [BDN+11] for more details. The objective is to have a fixed input-length MAC. Suppose that C is a secret and U is a one block challenge. The value V_T is the response or the authentication tag. We write

$$V_T = \mathsf{ARMADILLO}(C, U)$$

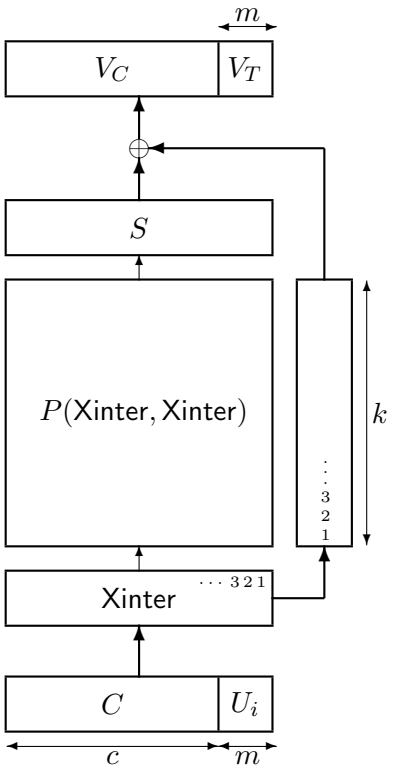


Figure 12.3: Scheme of ARMADILLO1b.

As can be seen from the description of the algorithm, there is no substitution layer. This means that for a fixed key C, the permutation σ_C is fixed (but unknown). As we see later in this chapter, it can be easily recovered. For the attack, it suffices to recover the mapping σ_C of a single index, for instance we recover $\sigma_C(j) = n$ for some value j. If we can recover the mapping $\sigma_C(j)$, we then take the challenges U_i so that j-th bit of $P(U_i, C\|U_i)$ contains different bits of the key. This allows us to recover the secret key from literally reading the key from the output of ARMADILLO1b. More precisely, we consider

$$T_1(X) = X$$
$$T_3(b, X) = X_{\sigma_b} \oplus \gamma$$

with arbitrary linear T_2 and T_4. This includes ARMADILLO1 and ARMADILLO1b.

The attack is based on the fact that a bit permutation is linear with respect to an XOR operation, i.e., for a permutation σ, X and Y be two vectors, we have $(X \oplus Y)_\sigma = X_\sigma \oplus Y_\sigma$.

Lemma 12.1. *For any T_3, C and U, we have*

$$P(C\|U, C\|U) = P(C, P(U, C\|U))$$

Proof. We easily prove it by the induction on the size of C. □

Lemma 12.2. *For $T_3(b, X) = X_{\sigma_b} \oplus \gamma$, there exists a function $f : 2^{|X|} \to 2^{|X|}$ such that for any $Y = (y_k\| \ldots \|y_1)$ and X, we have*

$$P(Y, X) = X_{\sigma_Y} \oplus f(Y)$$

Proof. Let's rewrite

$$P(Y,X) = \Big(\Big(\big(X_{\sigma_{y_1}} \oplus \gamma\big)_{\sigma_{y_2}} \oplus \gamma\Big)_{\sigma_{y_3}} \oplus \gamma \dots\Big)_{\sigma_{y_k}} \oplus \gamma$$

Let's define the *prefix* of Y as

$$\text{prefix}(Y) = \{Y_j;\ Y_j = (y_k \| \dots \| y_j), 1 \le j \le k\}$$

Thus, P can be rewritten as

$$P(Y,X) = (X \oplus \gamma)_{\sigma_Y} \oplus \gamma \oplus \bigoplus_{p \in \text{prefix}(Y)} \gamma_{\sigma_p} = X_{\sigma_Y} \oplus P(Y,0)$$

□

Now we apply the above results to ARMADILLOgen with $T_1(X) = X$ and $T_3(b,X) = X_{\sigma_b} \oplus \gamma$.

$$\begin{aligned}
\mathsf{ARMADILLOgen}(C\|U) &= T_4(P(C\|U, T_2(C\|U)), C\|U) \\
&= T_4(P(C, P(U, T_2(C\|U))), C\|U) \\
&= T_4(P(C, (L_U(C)_{\sigma_U} \oplus f(U))), C\|U) \\
&= T_4\left((L_U(C)_{\sigma_U} \oplus f(U))_{\sigma_C} \oplus f(C), C\|U\right)
\end{aligned}$$

where $L_U(C) = T_2(C\|U)$ and $f(U)$ is given by Lemma 12.2. The first equality is coming from the definition, the second from Lemma 12.1 and the last two from Lemma 12.2. So, we can write

$$\mathsf{ARMADILLOgen}(C\|U) = L\left((L_U(C)_{\sigma_U} \oplus f(U))_{\sigma_C} \oplus g(U) \oplus h(C)\right)$$

for some linear function L and some functions g and h. For all the variants we consider, L is either the identity function or consists of dropping a few bits. For ARMADILLO1b and ARMADILLO1 the function $h(C) = f(C) \oplus (C\|0^m)$, $g(U) = (0^c\|U)$. Similarly, $L(X) = X$ and $L(X) = \mathsf{tail}_k(X)$ respectively.

In what follows, we consider an arbitrary i and take a vector e_i such that $e_i \cdot L(X) = X[i]$, i.e., the i-th bit of register X. So, we obtain

$$e_i \cdot \mathsf{ARMADILLOgen}(C\|U) = (L_U(C)_{\sigma_U} \oplus f(U))_{\sigma_C^{-1}(i)} \oplus g(U)_i \oplus h(C)_i$$

Clearly, there exists a $j = \sigma_C^{-1}(i)$ such that

$$e_i \cdot \mathsf{ARMADILLOgen}(C\|U) \oplus g(U)_i = L_U(C)_{\sigma_{U_t}^{-1}(j)} \oplus f(U)_j \oplus h(C)_i \tag{12.1}$$

In chosen-input attacks against the PRF mode, we assume that the adversary can compute

$$e_i \cdot \mathsf{ARMADILLOgen}(C\|U)$$

for a chosen U and a secret C. In the challenge-response application, we only have access to V_T, but in all considered variants, e_i has Hamming weight one, so we just need to select i so that this bit lies in the V_T window. We introduce an attack (see Algorithm 12.1) which only needs this bit of the response for $n = k \log k$ queries. This algorithm has complexity $\mathcal{O}(k^2 \log k)$ to recover the secret C

(also see Figure 12.4). In fact, the attacker can simply recover the permutation $Y = P(U_i, \mathsf{Xinter})$, as she has control over U_i's. Now, her goal is to find out how $P(C, Y)$ maps the index j to i. The goal of the algorithm is to find this mapping and to recover C. It is exploiting the fact that fixing i, then $h(C)_i$ is fixed for all challenges and the left side of Eq. (12.1) can be computed directly by the adversary. Then, it recovers C by solving an overdefined linear system of equations and check whether it has a solution. If so, it checks whether the recovered C is consistent with other samples.

Algorithm 12.1 The key recovery algorithm against ARMADILLO1 and ARMADILLO1b.

1: Pick a random i from 1 to m.
2: **for** t from 1 to $n = k \log k$ **do**
3: collect a challenge-response pair $(U_t, e_i \cdot \mathsf{ARMADILLOgen}(C \| U_t))$
4: compute $b_t = e_i \cdot \mathsf{ARMADILLOgen}(C \| U_t) \oplus g(U_t)_i$.
5: **end for**
6: **for** j from 1 to ℓ **do**
7: **for** each $\beta \in \{0, 1\}$ **do**
8: set $h(C)_i = \beta$.
9: **for** t from 1 to n **do**
10: compute $L_{U_t}(C)_{\sigma_{U_t}^{-1}(j)} = b_t \oplus f(U_t)_j \oplus \beta$ for all c bits.
11: **end for**
12: solve the system of n linear equations $L_{U_t}(C)_{\sigma_{U_t}^{-1}(j)}$
13: **if** no solution **then**
14: **break**
15: **end if**
16: derive C
17: **if** C is consistent with samples **then**
18: output C.
19: **end if**
20: **end for**
21: **end for**

The Attack complexity. The first **for** loop runs ARMADILLO algorithm $k \log k$ times . The second loop runs ℓ times, where $\ell = 2k$ for ARMADILLO1 and $\ell = k$ for ARMADILLO1b. We perform up to $2k \log k$ simple arithmetic operations in the second loop to compute the values $L_{U_t}(C)_{\sigma_{U_t}^{-1}(j)}$. Solving the system of n linear equation requires $O(n^3)$ in general case. However, in the case of ARMADILLO1 and ARMADILLO1b, every line contains only one variable of secret C, which comes from Lemma 12.2. As we have $k \log k$ equations in c variables, if the mapping $i \to j$ is not guessed correctly, we have a high probability to obtain a contradiction in line 13. So overall, we have complexity of $O(k^2 \log k)$ for attacking both ARMADILLO1 and ARMADILLO1b.

The Probability of success. We first choose $k \log k$ challenges U_t randomly and compute the function $\mathsf{ARMADILLOgen}(C \| U_t)$. That is because according to *coupon collector problem* [KJV07], the expected number of challenges so that every bit of C is mapped to the i-th bit of the output is $k \log k$. Therefore, among $k \log k$ challenges all the bits of the challenge and all the bits of the secret key are mapped to a single bit of the output. The attacker can derive an equation for the j-th bit of $P(U_t, C \| U_t)$ and for $k \log k$ distinct challenges U_t the set of equations will have full

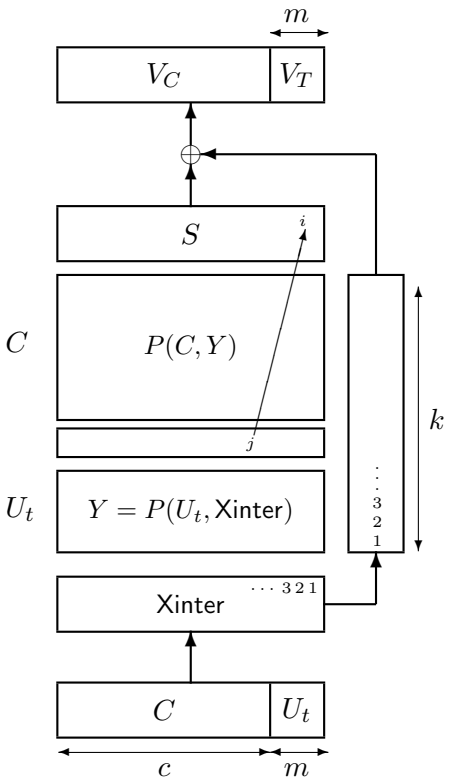


Figure 12.4: Scheme of the key recovery algorithm against ARMADILLO1 and ARMADILLO1b.

rank. These equations do not change through the fixed mapping σ_C, only the constant term might change due to the term $P(C, 0)$. Therefore, if the attacker guess $j \to i$ correctly, the set of $k \log k$ equations in c variables has a solution, otherwise the set of $k \log k$ equations in c variables has no solution with probability at least $1 - 2^{-n}$.

Chapter 13

Conclusion

We shed some light on the theory behind the ElimLin algorithm. This yielded a better understanding of how this algorithm works. But, there is still a lot of work to be done to evaluate the security of the ciphers against this simple algorithm. Then, we evaluated the security of several block ciphers using ElimLin. We showed that it might not be a fair measure to compare the lightweight block ciphers with respect to the number of gates, and throughput when they do not provide the same level of security encountering various types of attacks. Next, we proposed a pre-processing technique applied to the algebraic representation of the ciphers before yielding it to a SAT solver. This pre-processing is somehow equivalent to running F4 for a while and then stopping and delivering the system to a SAT solver. We finally proposed a practical attack on the multi-purpose cryptographic primitive ARMADILLO1. We showed that the key can be recovered in a matter of seconds.

Bibliography

[AA09] F. Armknecht and G. Ars. Algebraic Attacks on Stream Ciphers with Gröbner Bases. *Gröbner Bases, Coding, and Cryptography*, pages 329–348, 2009.

[ABNP+11] M. Abdelraheem, C. Blondeau, M. Naya-Plasencia, M. Videau, and E. Zenner. Cryptanalysis of ARMADILLO2. In *ASIACRYPT*, volume 7073, pages 308–326. Springer, 2011.

[Ans50] F.J. Anscombe. Sampling theory of the negative binomial and logarithmic series distributions. *Biometrika*, 37(3-4):358–382, 1950.

[Bar09] G.V. Bard. *Algebraic Cryptanalysis*. Springer, 2009.

[BC08] E. Biham and Y. Carmeli. Efficient Reconstruction of RC4 Keys from Internal States. In *FSE*, volume 5086, pages 270–288. Springer, 2008.

[BCJ07] G. Bard, N. Courtois, and C. Jefferson. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. In *presented at ECRYPT workshop Tools for Cryptanalysis*, 2007. `http://eprint.iacr.org/2007/024.pdf`.

[BCN+10] G. Bard, N. Courtois, J. Nakahara, P. Sepehrdad, and B. Zhang. Algebraic, AIDA/Cube and Side Channel Analysis of KATAN Family of Block Ciphers. In *INDOCRYPT*, volume 6498, pages 176–196. Springer, 2010.

[BD07] M. Brickenstein and A. Dreyer. PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials. In *Electronic Proceedings of MEGA 2007*, 2007. `http://www.ricam.oeaw.ac.at/mega2007/electronic/26.pdf`.

[BDN+11] S. Badel, N. Dağtekin, J. Nakahara, Kh. Ouafi, N. Reffé, P. Sepehrdad, P. Sušil, and S. Vaudenay. ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In *CHES*, volume 6225, pages 398–412. Springer, 2011.

[BF53] C.I. Bliss and R.A. Fisher. Fitting the Negative Binomial Distribution to Biological Data. *Biometrika*, 9:176–200, 1953.

[BGP06] C. Berbain, H. Gilbert, and J. Patarin. QUAD: A practical stream cipher with provable security. In *EUROCRYPT*, volume 4004, pages 109–128. Springer, 2006.

[BGW01] N. Borisov, I. Goldberg, and D. Wagner. Intercepting mobile communications: the insecurity of 802.11. In *MobiCom*, pages 180–189. ACM, 2001.

[Bit03] A. Bittau. Additional Weak IV Classes for the FMS Attack, 2003. `http://www.netstumbler.org/showthread.php?postid=89036#pos%t89036`.

[BKL+07] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: an Ultra-Lightweight Block Cipher. In *CHES*, volume 4727, pages 450–466. Springer, 2007.

[BT09] M. Beck and E. Tews. Practical Attacks Against WEP and WPA. In *WISEC*, pages 79–86. ACM, 2009.

[Buc06] B. Buchberger. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of Symbolic Computation*, 41(3-4):475–511, 2006.

[CB07] N. Courtois and G.V. Bard. Algebraic Cryptanalysis of the Data Encryption Standard. In *IMA Int. Conf.*, volume 4887, pages 152–169. Springer, 2007.

[CD08] N. Courtois and B. Debraize. Algebraic Description and Simultaneous Linear Approximations of Addition in Snow 2.0. In *ICICS*, volume 5308, pages 328–344. Springer, 2008.

[Cha06] R. Chaabouni. Breaking WEP Faster with Statistical Analysis. Technical report, EPFL/LASEC, 2006. Semester project.

[CM03] N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - EUROCRYPT*, volume 2656, pages 345–359. Springer, 2003.

[COQ09] N. Courtois, S. O'Neil, and J. Quisquater. Practical Algebraic Attacks on the Hitag2 Stream Cipher. In *ISC*, volume 5735, pages 167–176, 2009.

[Cou02] N. Courtois. Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt. In *ICISC*, volume 2587, pages 182–199, 2002.

[Cou03] N. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - CRYPTO*, volume 2729, pages 176–194. Springer, 2003.

[Cou08] N. Courtois. Tools for Experimental Algebraic Cryptanalysis, 2008. `http://www.cryptosystem.net/aes/tools.html`.

[Cou09] N. Courtois. The Dark Side of Security by Obscurity - and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime. In *SECRYPT*, pages 331–338, 2009.

[Cou11] N. Courtois. Algebraic Complexity Reduction and Cryptanalysis of GOST. In *Cryptology ePrint Archive*, 2011. `http://eprint.iacr.org/2011/626`.

[CP02] N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *Asiacrypt*, volume 2501, pages 267–287. Springer, 2002.

[CSPK00] N. Courtois, A. Shamir, J. Patarin, and A. Klimov. Efficient Algorithms for Solving Overfined Systems of Multivariate Polynomial Equations. In *Advances in Cryptology, Eurocrypt*, volume 1807, pages 392–407. Springer, 2000.

[Dav73] R.B. Davies. Numerical inversion of a characteristic function. *Biometrika*, 60(2):415–417, 1973.

[Dav80a] R.B. Davies. The distribution of a linear combination of chi-squared random variables. *Applied Statistics*, 29:323–333, 1980.

[Dav80b] R.B. Davies. Linear combination of chi-squared random variables, 1980. `http://www.statsresearch.co.nz/robert/QF.htm`.

[DCDK09] C. De Canniére, O. Dunkelman, and M. Knezević. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In *CHES*, volume 5747, pages 272–288, 2009.

[DCP08] C. De Canniére and B. Preneel. Trivium. in New Stream Cipher Designs. In *The eSTREAM Finalists*, volume 4986, pages 84–97. Springer, 2008.

[DO11] C. Devine and T. Otreppe. Aircrack-ng, accessed October 22, 2011. `http://www.aircrack-ng.org/`.

[DS09] I. Dinur and A. Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *Advances in Cryptology - EUROCRYPT*, volume 5479, pages 278–299. Springer, 2009.

[DS11] I. Dinur and A. Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In *FSE*, volume 6733, pages 167–187. Springer, 2011.

[ES] N. Eén and N. Sörensson. MiniSat 2.0. An open-source SAT solver package. `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/`.

[ES05] N. Een and N. Sorensson. MiniSat - A SAT Solver with Conflict-Clause Minimization. In *Theory and Applications of Satisfiability Testing*, 2005.

[Fau99] J. Faugère. A new effcient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.

[Fau02] J. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *Symbolic and Algebraic Computation - ISSAC*, page 75–83, 2002.

[FB09] G. Fusco and E. Bach. Phase transition of multivariate polynomial systems. *Journal of Mathematical Structures in Computer Science*, 19(1), 2009.

[Fel43] W. Feller. On a general class of "contagious" distributions. *Ann. Math. Stat.*, 14:389–400, 1943.

[Fer02] N. Ferguson. fel: an Improved MIC for 802.11 WEP. IEEE doc. 802.11-2/020r0, 2002.

[FM01] S.R. Fluhrer and D.A. McGrew. Statistical Analysis of the Alleged RC4 Keystream Generator. In *FSE*, volume 1978, pages 19–30. Springer, 2001.

[FMS01] S.R. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In *SAC*, volume 2259, pages 1–24. Springer, 2001.

[Gha05] M. Ghasemzadeh. *A New Algorithm for the Quantified Satisfiability Problem, Based on Zero-suppressed Binary Decision Diagrams and Memoization.* PhD thesis, University of Potsdam, Germany, 2005.

[Gol97] J.Dj. Golic. Linear Statistical Weakness of Alleged RC4 Keystream Generator. In *EUROCRYPT*, volume 1233, pages 226–238. Springer, 1997.

[Gol00] J.Dj. Golic. Iterative Probabilistic Cryptanalysis of RC4 Keystream Generator. In *ACISP*, volume 1841, pages 220–223. Springer, 2000.

[GP51] J. Gil-Pelaez. Note on the inversion theorem. *Biometrika*, 38(3/4):481–482, 1951.

[Hul] D. Hulton. bsd-airtools. `http://www.dachb0den.com/projects/bsd-airtools.html`.

[Hul01] D. Hulton. Practical Exploitation of RC4 Weaknesses in WEP Environments, 2001. `http://www.dartmouth.edu/~madory/RC4/wepexp.txt`.

[HWF02] R. Housley, D. Whiting, and N. Ferguson. Alternate Temporal Key Hash. IEEE doc. 802.11-02/282r2, 2002.

[IEE99a] IEEE. ANSI/IEEE Standard 802.11b: Wireless LAN Medium Access Control (MAC) and Physical Layer (phy) Specification, 1999.

[IEE99b] IEEE. IEEE Std 802.11, Standards for Local and Metropolitan Area Networks: Wireless Lan Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999.

[IEE03] IEEE. ANSI/IEEE standard 802.11i, Amendment 6 Wireless LAN Medium Access Control (MAC) and Physical Layer (phy) Specifications, 2003. Draft 3.

[IKD+08] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on Keeloq. In *EUROCRYPT*, volume 4965, pages 1–18, 2008.

[Imh61] J.P. Imhoff. Computing the distribution of quadratic forms in normal variables. *Biometrika*, 48:419–426, 1961.

[Jen96] R. Jenkins. ISAAC and RC4, 1996. `http://burtleburtle.net/bob/rand/isaac.html`.

[JK70] N.L. Johnson and S. Kotz. *Continuous Univariate Distributions.* Houghton Mifflin, 1970.

[Jun03] P. Junod. On the Optimality of Linear, Differential, and Sequential Distinguishers. In *EUROCRYPT*, volume 2656, pages 255–271. Springer, 2003.

[JV03] P. Junod and S. Vaudenay. Optimal Key Ranking Procedures in a Statistical Cryptanalysis. In *FSE*, volume 2656, pages 235–246. Springer, 2003.

[KJV07] J. Kobza, S. Jacobson, and D. Vaughan. A Survey of the Coupon Collector's Problem with Random Sample Sizes. *Methodology and Computing in Applied Probability*, 9(4):573–584, 2007.

[Kle08] A. Klein. Attacks on the RC4 Stream Cipher. *Design, Codes, and Cryptography*, 48:269–286, 2008.

[KMP+98] L.R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege. Analysis Methods for (Alleged) RC4. In *ASIACRYPT*, volume 1514, pages 327–341. Springer, 1998.

[Kor04a] Korek. Need Security Pointers, 2004. `http://www.netstumbler.org/showthread.php?postid=89036#pos%t89036`.

[Kor04b] Korek. Next Generation of WEP Attacks?, 2004. `http://www.netstumbler.org/showpost.php?p=93942&postcount=%35`.

[Laz83] D. Lazard. Gröbner-bases, Gaussian elimination and resolution of systems of algebraic equations. In *European Computer Algebra Conference on Computer Algebra*, volume 162. Springer, 1983.

[Mac16] F.S. Macaulay. *The algebraic theory of modular systems.* Cambridge Mathematical Library, 1916.

[Mag] Magma, software package. `http://magma.maths.usyd.edu.au/magma/`.

[Man01] I. Mantin. Analysis of the Stream Cipher RC4. Master's thesis, Weizmann Institute of Science, 2001.

[Man05a] I. Mantin. A Practical Attack on the Fixed RC4 in the WEP Mode. In *ASIACRYPT*, volume 3788, pages 395–411. Springer, 2005.

[Man05b] I. Mantin. Predicting and Distinguishing Attacks on RC4 Keystream Generator. In *EUROCRYPT*, volume 3494, pages 491–506. Springer, 2005.

[Mar57] H.M. Markovitz. The Elimination Form of the Inverse and Its Application to Linear Programming. *Management Science*, pages 225–269, 1957.

[Max05] A. Maximov. Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness. In *FSE*, volume 3557, pages 342–358. Springer, 2005.

[MD95] S. Murphy and D. Davies. Pairs and triples of DES S-boxes. *Journal of Cryptology*, 8:1–25, 1995.

[Mir02] I. Mironov. Not So Random Shuffles of RC4. In *CRYPTO*, volume 2442, pages 304–319. Springer, 2002.

[MK08] A. Maximov and D. Khovratovich. New State Recovery Attack on RC4. In *CRYPTO*, volume 5157, pages 297–316. Springer, 2008.

[MP08] S. Maitra and G. Paul. New Form of Permutation Bias and Secret Key Leakage in Keystream Bytes of RC4. In *FSE*, volume 5086, pages 253–269. Springer, 2008.

[MR02] S. Murphy and M. Robshaw. Essential Algebraic Structure within AES. In *Advances in Cryptology - CRYPTO*, volume 2442, pages 1–16. Springer-Verlag, 2002.

[MRH04] V. Moen, H. Raddum, and K.J. Hole. Weaknesses in the Temporal Key Hash of WPA. *Mobile Computing and Communications Review*, 8:76–83, 2004.

[MS01] I. Mantin and A. Shamir. A Practical Attack on Broadcast RC4. In *FSE*, volume 2355, pages 152–164. Springer, 2001.

[Mui05] R. Muirhead. *Aspects of Multivariate Statistical Theory*. Wiley, 2005.

[Ney39] J. Neyman. On a new class of "contagious" distributions, applicable in entomology and bacteriology. *Ann. Math. Stat.*, 10:35–57, 1939.

[NPP12] M. Naya-Plasencia and T. Peyrin. Practical Cryptanalysis of ARMADILLO2. In *FSE*. Springer, 2012.

[NS09] K. Nohl and M. Soos. Solving Low-Complexity Ciphers with Optimized SAT Solvers. In *EUROCRYPT*, 2009.

[NSZW09] J. Nakahara, P. Sepehrdad, B. Zhang, and M. Wang. Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT. In *CANS*, volume 5888, pages 58–75. Springer, 2009.

[NW06] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer Verlag, second edition, 2006.

[Ori] ORIDAO. `http://www.oridao.com/`.

[PM07] G. Paul and S. Maitra. Permutation After RC4 Key Scheduling Reveals the Secret. In *SAC*, volume 4876, pages 360–377. Springer, 2007.

[PP04] S. Paul and B. Preneel. A New Weakness in the RC4 Keystream Generator and an Approach. In *FSE*, volume 3017, pages 245–259. Springer, 2004.

[PR88] J. Postel and J. Reynolds. A Standard for the Transmission of IP Datagrams over IEEE 802 Networks, 1988.
`http://www.cs.berkeley.edu/$\sim$daw/my-posts/my-rc4-weak-keys`.

[PRM08] 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. *Design, Codes, and Cryptography*, 49:123–134, 2008.

[Roo95] A. Roos. A Class of Weak Keys in RC4 Stream Cipher (sci.crypt), 1995.
`http://marcel.wanda.ch/Archive/WeakKeys`.

[RS08] H. Raddum and I. Semaev. Solving Multiple Right Hand Sides linear equations. *Journal of Designs, Codes and Cryptography*, 49(1-3):147–160, 2008.

[SAT] SAT. SAT Race Competition. `http://www.satcompetition.org/`.

[Sep08] P. Sepehrdad. Algebraic cryptanalysis of ciphers and elimination methods. Master's thesis, University College London, 2008.

[SGMPS11a] S. Sen Gupta, S. Maitra, G. Paul, and S. Sarkar. Proof of Empirical RC4 Biases and New Key Correlations. In *SAC*, volume 7118, pages 151–168. Springer, 2011.

[SGMPS11b] S. Sen Gupta, S. Maitra, G. Paul, and S. Sarkar. RC4: (Non-)Random Words from (Non-)Random Permutations. In *Cryptology ePrint Archive*, 2011.
`http://eprint.iacr.org/2011/448.pdf`.

[Sha49] C.E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28, 1949.

[Sie85] D. Siegmund. *Sequential analysis - tests and confidence intervals*. Springer, 1985.

[Sin99] S. Singh. *The code book: the evolution of secrecy from Mary, Queen of Scots, to quantum cryptography*. Jeffery Design, 1999.

[SIR02] A. Stubblefield, J. Ioannidis, and A.D. Rubin. Using the Fluhrer, Mantin, and Shamir Attack to Break WEP. *Network and Distributed System Security Symposium (NDSS)*, 2002.

[SIR04] A. Stubblefield, J. Ioannidis, and A.D. Rubin. A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP). *ACM Transactions on Information and System Security (TISSEC)*, 7(2), 2004.

[SPSG11] M. Subhamoy, G. Paul, and S. Sen Gupta. Attack on Broadcast RC4 Revisited. In *FSE*, volume 6733, pages 199–217. Springer, 2011.

[SSV11] P. Sepehrdad, P. Sušil, and S. Vaudenay. Fast Key Recovery Attack on ARMADILLO1 and Variants. In *CARDIS*, volume 7079, pages 133–150, 2011.

[SSV12] P. Sepehrdad, P. Sušil, and S. Vaudenay. Elimlin Algorithm Revisited. In *FSE*, 2012.

[Ste] W. Stein. SAGE, software package. `http://www.sagemath.org/`.

[Stu07] Student. On the error of counting with a haemocytometer. *Biometrika*, 5:351–360, 1907.

[SVV10] P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Discovery and Exploitation of New Biases in RC4. In *SAC*, volume 6544, pages 74–91. Springer, 2010.

[SVV11] P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Statistical Attack on RC4: Distinguishing WPA. In *EUROCRYPT*, volume 6632, pages 343–363. Springer, 2011.

[SVV12] P. Sepehrdad, M. Vuagnoux, and S. Vaudenay. Tornado Attack on RC4 with Applications to WEP and WPA. *Submitted to Journal of Cryptology*, 2012.

[TBNT07] V. Tomasevic, S. Bojanic, and O. Nieto-Taladriz. Finding an Internal State of RC4 Stream Cipher. *Information Sciences: an International Journal*, 177:1715–1727, 2007.

[Tho57] H.C.S. Thom. The Frequency of Hail Occurrence. *Theoretical and Applied Climatology*, 8:185–194, 1957.

[Tho63] H.C.S. Thom. Tornado Probabilities. *American Meteorological Society*, pages 730–736, 1963.

[TWP07] E. Tews, R. Weinmann, and A. Pyshkin. Breaking 104 Bit WEP in Less Than 60 Seconds. In *WISA*, volume 4867, pages 188–202. Springer, 2007.

[Vie07] M. Vielhaber. Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. In *Cryptology ePrint Archive*, 2007. `http://eprint.iacr.org/2007/413`.

[Vua10] M. Vuagnoux. *Computer Aided Cryptanalysis from Ciphers to Side Channels*. PhD thesis, EPFL, Switzerland, 2010.

[VV07] S. Vaudenay and M. Vuagnoux. Passive-only Key Recovery Attacks on RC4. In *SAC*, volume 4876, pages 344–359. Springer, 2007.

[Wag95] D. Wagner. Weak Keys in Rc4 (sci.crypt), 1995. `http://www.cs.berkeley.edu/$\sim$daw/my-posts/my-rc4-weak-keys`.

[WB10] K.K.H Wong and G. Bard. Improved Algebraic Cryptanalysis of QUAD, Bivium and Trivium via Graph Partitioning on Equation Systems. In *ACISP*, 2010.

[Wei03] R. Weinmann. Evaluating Algebraic Attacks on the AES. Master's thesis, Technische Universität Darmstadt, 2003.

[WG.01] IEEE 802.1 WG. 802.1x: Standards for Local and Metropolitan Area Networks: Port-Based Access Control, 2001. Draft 3.

[Whi14] L. Whitaker. On the Poisson law of small numbers. *Biometrika*, 10:36–71, 1914.

[WZ11] W. Wu and L. Zhang. LBlock: A Lightweight Block Cipher. In *ACNS*, volume 6715, pages 327–344. Springer, 2011.

Appendix A

Probability Distributions

A.1 Quadratic Forms in Normal Random Variables

The cumulative distribution function (cdf) of standard normal distribution is represented as φ function and is defined as

$$\varphi(\lambda) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\lambda} e^{-\frac{x^2}{2}}\, dx = \frac{1}{2}\mathsf{erfc}\left(-\frac{\lambda}{\sqrt{2}}\right)$$

In particular, $\varphi(-\lambda/\sqrt{2}) = \frac{1}{2}\mathsf{erfc}(\frac{\lambda}{2})$.

Let X_i's be r independent, normally distributed random variables with means μ_i's and variances σ_i^2's. Then, the random variable

$$T = \sum_{i=1}^{r} \left(\frac{X_i}{\sigma_i}\right)^2$$

is distributed according to the non-central χ^2 distribution. It has two parameters: r which specifies the number of degrees of freedom (i.e. the number of X_i's), and λ which is related to the mean of the random variables X_i's by

$$\lambda = \sum_{i=1}^{r} \left(\frac{\mu_i}{\sigma_i}\right)^2$$

λ is called the noncentrality parameter.

Note. For the probability density function of non-central χ^2 distribution, see any book on probability.

A quadratic form in independent normal random variables can be expressed as the linear combination

$$Q = \sum_{i=1}^{k} a_i X_i + a_0 X_0$$

where X_i's are independent and follow non-central χ^2 distribution with n_i degrees of freedom and non-centrality parameter α_i^2 for $i \in \{1, \ldots, k\}$ and X_0 has a standard normal distribution. There is no closed formula for the density function of this distribution, but there are several methods to compute the distribution of Q (see Chapter 29 of [JK70] for instance). A numerical method

was also suggested by Imhoff [Imh61]. Programs in Fortran and in C were written by Robert Davies [Dav80b] to compute this probability distribution numerically. We call the distribution of Q the Generalized χ^2 distribution. The characteristic function of this distribution is given by

$$\varphi_Q(u) = E(e^{iuQ}) = \frac{e^{iu\left(\frac{\sum_{j=1}^{k} a_j \alpha_j^2}{1-2iua_j}\right) - \frac{1}{2}u^2 a_0^2}}{\prod_{j=1}^{k}(1-2iua_j)^{\frac{1}{2}n_j}}$$

If $E(Q)$ is finite, following Gil-Pelaez [GP51] the cumulative distribution function (cdf) of this random variable can be computed using the characteristic function as

$$F_Q(q) = \mathsf{Pr}(Q < q) = \frac{1}{2} - \int_{-\infty}^{\infty} \mathsf{Im}\left(\frac{\varphi_Q(u)e^{-iux}}{2\pi u}\right) du$$

A.2 Some Further Probability Distributions and Functions

Gamma Function. The gamma function over complex numbers is an extension of the factorial function and is defined as

$$\Gamma(x) = \int_0^{\infty} t^{x-1}e^{-t}dt$$

for $\mathsf{Re}(x) > 0$.

Beta Function. The beta function, also called the Euler integral of the first kind, over complex numbers is defined by

$$B(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1}dt$$

for $\mathsf{Re}(a) > 0$ and $\mathsf{Re}(b) > 0$.

Incomplete Beta Function. The incomplete beta function is a generalization of the beta function and is defined as

$$B(x;a,b) = \int_0^x t^{a-1}(1-t)^{b-1}dt$$

Regularized Incomplete Beta Function. The regularized incomplete beta function is defined in terms of the incomplete beta function and the complete beta function as

$$I_x(a,b) = \frac{B(x;a,b)}{B(a,b)}$$

Negative Binomial Distribution. We say that X has a negative binomial distribution if it has probability mass function:

$$\mathsf{Pr}[X = x] = \binom{x+r-1}{x}(1-p)^r p^x$$

where r is a positive integer and p is real.

r and p are both parameters of this distribution. Extending this definition by letting r to be real positive, the binomial coefficient is then defined by the multiplicative formula and can also be rewritten using the gamma function:

$$\mathsf{Pr}[X = x] = \frac{\Gamma(x + r)}{x!\Gamma(r)}(1 - p)^r p^x$$

This generalized distribution is called the Pólya distribution. We also have

$$E(X) = \frac{pr}{(1 - p)} \qquad \text{and} \qquad V(X) = \frac{pr}{(1 - p)^2}$$

The cdf of this distribution can be computed using regularized incomplete beta function. In fact, we have

$$F_X(x) = \mathsf{Pr}(X \leq x) = 1 - I_p(x + 1, r)$$

Curriculum Vitæ

PERSONAL PROFILE

Name: Pouyan Sepehrdad
Gender: Male
Date of Birth: 6th Sep, 1984
Nationality: Iranian
Marital Status: Single
Languages: Persian, English

EDUCATION

École Polytechnique Fédérale de Lausanne (EPFL); PhD in Computer Science (Cryptography) at LASEC laboratory (2008-2012) [Lausanne, Switzerland]

University College London (UCL); Msc. in Information Security [Computer Security track, specialized in Cryptography] (2007-2008) [London, UK]

Isfahan University of Technology (IUT); Bsc. in Electrical Engineering [specialized in Electronics] (2003-2007) [Isfahan, Iran]

Imam Mohammad-Bagher Pre-University School; Pre-University degree in Mathematics / Physics (2002-2003) [Isfahan, Iran]

Imam Mohammad-Bagher High School; Diploma in Mathematics / Physics (1998-2002) [Isfahan, Iran]

REVIEWED PUBLICATIONS

1. N. Courtois, P. Sepehrdad, P. Sušil, and S. Vaudenay. ElimLin Algorithm Revisited. In FSE'12.
2. M. Sajadieh, M. Dakhilalian, H. Mala, and P. Sepehrdad. Recursive Diffusion Layers for Block Ciphers and Hash Functions. In FSE'12.
3. P. Sepehrdad, P. Sušil, and S. Vaudenay. Fast Key Recovery Attack on ARMADILLO1 and Variants. In CARDIS'11, volume 7079, pages 133–150. Springer-Verlag, 2011.
4. P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Statistical Attack on RC4 - Distinguishing WPA. In EUROCRYPT'11, volume 6632, pages 343–363. Springer-Verlag, 2011.
5. S. Badel, N. Dağtekin, J. Nakahara, K. Ouafi, N. Reffé, P. Sepehrdad, P. Sušil, and S. Vaudenay. ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In CHES'10, volume 6225, pages 398–412. Springer-Verlag, 2010.
6. G.V. Bard, N. Courtois, J. Nakahara, P. Sepehrdad, and B. Zhang. Algebraic, AIDA/Cube and Side Channel Analysis of KATAN Family of Block Ciphers. In INDOCRYPT'10, volume 6498, pages 176–196. Springer-Verlag, 2010.
7. T. Baignères, P. Sepehrdad, and S. Vaudenay. Distinguishing Distributions Using Chernoff Information. In ProvSec'10, volume 6402, pages 144–165. Springer-Verlag, 2010.
8. P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. New Key Recovery Attacks on RC4/WEP. In 27th Chaos Communication Congress (CCC), 2010.
9. P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Discovery and Exploitation of New Biases in RC4. In SAC'10, volume 6544, pages 74–91. Springer-Verlag, 2010.
10. J. Nakahara, P. Sepehrdad, B. Zhang, and M. Wang. Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT. In CANS'09, volume 5888, pages 58–75. Springer-Verlag, 2009.
11. J. Aumasson, J. Nakahara, and P. Sepehrdad. Cryptanalysis of the ISDB Scrambling Algorithm (MULTI2). In FSE'09, volume 5665, pages 296–307. Springer-Verlag, 2009.
12. P. Sepehrdad. Algebraic Cryptanalysis of Ciphers and Elimination Methods. Master thesis, University College London, 2009.

SUBMITTED PUBLICATIONS

1. P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Tornado Attack on RC4 with Applications to WEP and WPA. (Submitted to Journal of Cryptology).
2. M. Sajadieh, M. Dakhilalian, H. Mala, and P. Sepehrdad. Efficient Recursive Diffusion Layers for Block Ciphers and Hash Functions. (Submitted to Journal of Cryptology).

EXPERIENCE

Research Assistant, LASEC-EPFL, Lausanne - 2008 - 2012
Research assistant and PhD student from Oct 2008 to June 2012 at Security and Cryptography Laboratory (LASEC) in Computer and Communication Sciences department (IC) at EPFL working under supervision of Prof. Serge Vaudenay.

IT Administration and Project Management, LASEC-EPFL, Lausanne - 2008 - 2012
The web administrator of LASEC laboratory at EPFL from Oct 2008 to June 2012; Also responsible for organizing, scheduling and assigning students' academic projects at LASEC; Supervised 8 master student projects during the period 2008-2012.

Integrated Circuit Designer , BehinehNiru Co. Esfahan, Iran - 2005-2006
Accomplished my Internship at Behinehniru company; Main responsibility: Integrated circuit design using FPGA programming with applications to power circuits; Designed an apparatus called "Power System Simulator" which was used for testing "Intelligent RTU, Power Analyzer and Fault & Event Recorder" manufactured by Behinehniru Co.

HONORS AND AWARDS

- Awarded one of the best paper awards (3 in total) of FSE 2012 for the paper: "Recursive Diffusion Layers for Block Ciphers and Hash Functions" (March 2012).
- Obtained Swiss National Science Foundation (SNSF) grant on a team project entitled "Cryptanalysis and Design of Ultra-lightweight Cryptographic Primitives" (March 2011 - March 2014).
- Awarded a fully-funded fellowship to study PhD in Computer Science at EPFL (Sep 2008 - Sep 2012).
- Distinction award from Computer Science department of University College London (UCL) following obtaining Msc. in Information Security with GPA: A.
- Awarded a fully-funded fellowship to study PhD in Computer Science by Computer Science department of University College London (UCL) [declined] (one of the only 4 scholarship positions available to overseas students in the entire department).
- Admission from Sharif University of Technology in Tehran to study master of science in Telecommunications specialized in Cryptography. [declined]
- Admission from Isfahan University of Technology in Isfahan to study bachelor of science in Electrical Engineering.
- Awarded the "Most Distinguished Student Award" at the second year of high school and "Top 3 Distinguished Students Award" at the first and third year of high school.

TECHNICAL SKILLS

- Windows, Linux, Mac OS X
- JAVA, C, C++, Python, VB.NET, Matlab, PHP
- Verilog, Quartus, MaxIIPlus, Active-HDL, PSpice, Orcad

TEACHING ACTIVITIES

Teaching assistant of the course "Cryptography and Security" during 3 consecutive years. (2009-2011)
Teaching assistant of the course "Advanced Cryptography" during 2 consecutive years. (2010-2011)

EXTERNAL REFERENCE

Acted as the external reviewer of various well-known cryptography and security journals and conferences such as Journal of Cryptology, ASIACRYPT, FSE, SAC and CT-RSA.