CRYPTANALYSIS OF SYMMETRIC KEY PRIMITIVES

ALEKSANDAR KIRCANSKI

A THESIS

IN

THE CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN INFORMATION SYSTEMS

SECURITY

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

JUNE 2009

Your file Votre référence
ISBN: 978-0-494-63043-3
Our file Notre référence
ISBN: 978-0-494-63043-3

ABSTRACT

Cryptanalysis of symmetric key primitives

Aleksandar Kircanski

Block ciphers and stream ciphers are essential building blocks that are used to construct computing systems which have to satisfy several security objectives. Since the security of these systems depends on the security of its parts, the analysis of these symmetric key primitives has been a goal of critical importance. In this thesis we provide cryptanalytic results for some recently proposed block and stream ciphers.

First, we consider two light-weight block ciphers, TREYFER and PIFEA-M. While TREYFER was designed to be very compact in order to fit into constrained environments such as smart cards and RFIDs, PIFEA-M was designed to be very fast in order to be used for the encryption of multimedia data. We provide a related-key attack on TREYFER which recovers the secret key given around 2^{11} encryptions and negligible computational effort. As for PIFEA-M, we provide evidence that it does not fulfill its design goal, which was to defend from certain implementation dependant differential attacks possible on previous versions of the cipher.

Next, we consider the NGG stream cipher, whose design is based on RC4 and aims to increase throughput by operating with 32-bit or 64-bit values instead of with 8-bit values.

We provide a distinguishing attack on NGG which requires just one keystream word. We also show that the first few kilobytes of the keystream may leak information about the secret key which allows the cryptanalyst to recover the secret key in an efficient way.

Finally, we consider GGHN, another RC4-like cipher that operates with 32-bit words. We assess different variants of GGHN-like algorithms with respect to weak states, in which all internal state words and output elements are even. Once GGHN is absorbed in such a weak state, the least significant bit of the plaintext words will be revealed only by looking at the ciphertext. By modelling the algorithm by a Markov chain and calculating the chain absorption time, we show that the average number of steps required by these algorithms to enter this weak state can be lower than expected at first glance and hence caution should be exercised when estimating this number.

Acknowledgments

I would like to acknowledge all the people who made this work possible.

First of all, I would like to express gratitude to my supervisor Dr. Amr Youssef for guiding me and unselfishly sharing his knowledge, enthusiasm and ideas with me. The amount of hours he spent considering the problems I encountered, his sincereness and openness made this work meaningful and possible.

I also wish to express gratitude to my parents. Without their beleif in my goals and their unselfish support, I would not be able to enter the field of scientific research.

Finally, I would like thank my wife Jelena for not urging me to find a job and supporting me to concentrate on cryptology. Also, I would like to thank Protee, for being such a great friend.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Historically, the classical goal of cryptography was to allow two persons to communicate over an insecure channel, such as telephone line, so that the adversary can not understand what is being said. Currently, the goal of cryptography, understood in a broader sense, is to provide means to enforce other security goals such as privacy, integrity, authenticity and non-repudiation.

The building blocks for ensuring these goals are crypto-primitives, such as block ciphers, stream ciphers, hash functions, and cryptographic protocols. An overview of these primitives is provided in Fig. 1. In what follows, we provide a brief introduction for these primitives. For further details, the reader is referred to [44].

- *Hash functions* are functions that take an arbitrary block of data and return a fixed size bit string. A secure hash function needs to be easy to compute and difficult to invert. Also, for any given message, it has to be infeasible to find another message with the same hash value. Finally, a secure hash function needs to be collision-free,

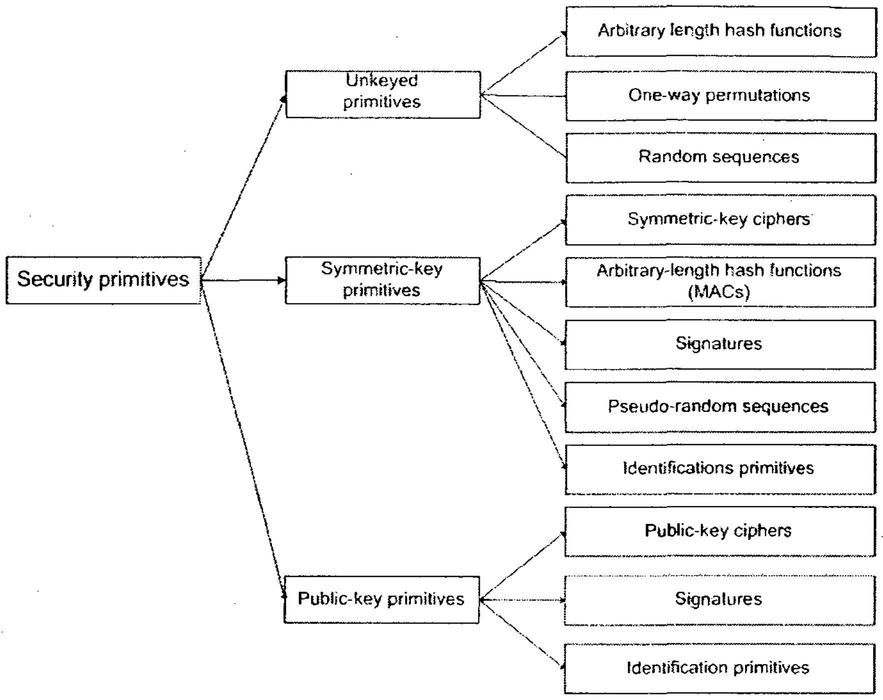

Figure 1: Classification of crypto-primitives [44]

i.e., it should be practically infeasible to find two different messages with the same hash value. Hash functions can be used as building blocks for ensuring integrity of information. Although in practice many constructions are believed to satisfy the above design objectives, no formal proofs exist to confirm this belief. MD5, SHA-1 and SHA-2 are some examples of commonly used hash functions. In response to recent cryptanalytic attacks on several hash functions, the National Institute for Standards and Technology (NIST) has opened a public competition to develop a new hash function. The new hash algorithm will be called SHA-3 and will augment the hash algorithms currently specified in FIPS 180-2.

- *One-way permutations*, like hash functions, need to be easy to compute but difficult to invert. The main difference is that unlike hash functions, one-way permutations are both injective and surjective. If proven to exist, these primitives can be used in the construction of several other primitives such as public-key cryptosystems, pseudorandom number generators, oblivious transfer protocols, and key agreement protocols.

- *Unkeyed cryptographically secure pseudo random generators* are typically used for secret key generation. For a secret key to be well chosen, it is necessary that the produced pseudorandom number sequence satisfies certain statistical properties such as balanced occurrence of 0's and 1's, and independence between numbers at different places in the sequence.

- *Symmetric key ciphers* include block ciphers and stream ciphers. By a block cipher we denote a function that transforms a plaintext to ciphertext and vice-versa, depending on the parameter called the secret key. Block ciphers operate on fixed-length groups of bits, called blocks, of typical sizes of 64-256 bits. Examples of block ciphers include the Advanced Encryption Standard (AES) [17] that has recently been approved by NIST as a replacement for the Data Encryption Standard (DES).

  A stream cipher is usually a pseudorandom number generator that depends on the secret key. The plaintext is then combined with the produced pseudorandom data, typically using XOR operation. Unlike the case for block ciphers, currently there is no specific standard for stream ciphers. On the other hand, the European Network of Excellence for Cryptology (ECRYPT), through its eSTREAM project that was finalized in September 2008, has identified a portfolio of seven promising new stream ciphers (HC-128, Grain, Rabbit, MICKEY, Salsa20/12, Trivium and SOSE-MANUK.)

- *Message Authentication Codes*, often called MACs, are short pieces of information used to authenticate data. MAC algorithms can be regarded as keyed hash functions. While similar to hash functions, MAC algorithms need to satisfy different security requirements. For instance, an adversary, given MACs of arbitrary number of messages computed under the secret key, should not be able to be able to deduce any information on the MAC of other messages.

- *Symmetric-key and public-key digital signatures* are used to give the receiver of the message assurance that the message was sent by claimed sender, even thought the channel of communication is insecure. They are implemented using symmetric-key primitives and public-key primitives. Digital signatures can also be used to enforce non-repudiation, according to which a signer of the message can not dispute that she is the actual signer of the message. The RSA, El-Gamal and NTRU signature algorithms are examples for public key digital signature schemes.

- *Symmetric key pseudorandom generators* typically include stream ciphers or a one-time pad system.

- *Identification systems* are used to allow one party (the verifier) to gain assurances that the identity of another (the claimant) is as declared, thus preventing impersonation.

The difference between identification systems and MACs is that message authentication does not provide interactive verification of identities. Identification systems are often built using other crypto-primitives such as public key ciphers or symmetric ciphers.

- *Public key ciphers*, also known as asymmetric key algorithms, use different keys for encryption and decryption. Each user has a pair of keys, a public and a private key. The private key is kept secret, whereas the public key can be publicly distributed. Messages are encrypted with user's public key and can only be decrypted by the user's private key. Although there exists a mathematical relation between the public and a private key, in order for the cryptosystem to be secure, it needs to be computationally infeasible to find the private key based on the knowledge of the public key. The RSA, El-Gamal and NTRU encryption algorithms are examples for public ciphers.

In this thesis, we focus on symmetric crypto-primitives, i.e., block and stream ciphers. In what follows, we provide a classification for both of these primitives and their corresponding definitions. A block cipher consists of two $1-1$ functions E_K, E_K^{-1}, both depending on the parameter K and mapping n-bit blocks to n-bit blocks. For each parameter K and n-bit string P

$$E_K^{-1}(E_K(P)) = P$$

The parameter K is called the secret key. P and $E_K(P)$ are referred to as the plaintext and ciphertext, respectively.

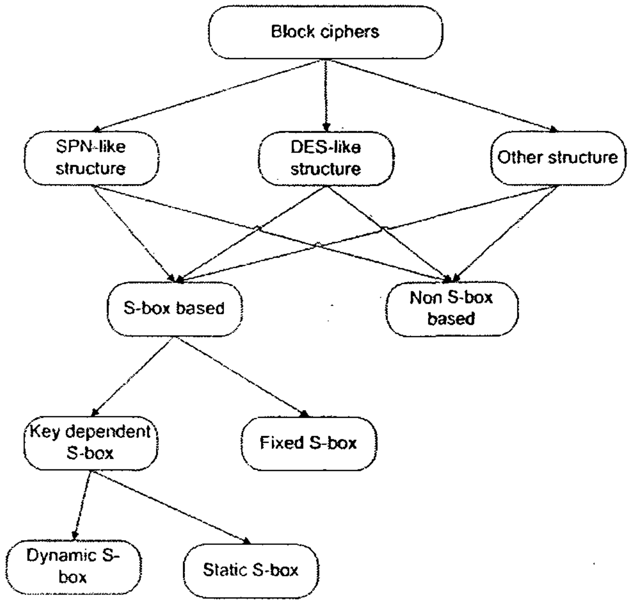

Figure 2: Classification of block ciphers

Construction of block ciphers is done by combining building blocks such as permutations, s-boxes and non-linear Boolean functions. Usually, these building blocks are combined in units called *rounds*. Fig. 2 shows a classification for block ciphers with respect to their construction. The substitution permutation network (SPN) structure and Fiestel (also referred to as DES-like) structures are the most commonly used designs for block ciphers constructions.

A typical round of an SPN-structure consists of nonlinear substitution operations to achieve the required confusion effect, linear transformation to achieve the required diffusion effect and key mixing operation. The user supplied key is often used to derive round keys using what is referred to as a key scheduling scheme. The AES [17] is an example of an SPN. As for DES-like structures, the plaintext is usually divided into two halves, denoted by L_i and R_i in round i. The following transformation is applied at the i-th round

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus F(R_i, K_i).$$

The round function, F, does not have to be $1-1$. The Data Encryption Standard (DES) is an example for a Feistel block cipher.

As shown in Fig. 2, some block ciphers achieve its required nonlinearity through the use of nonlinear s-boxes which are typically implemented as look up tables. When the the ciphers are not based on s-boxes, some other means of non-linearity must be introduced. This can include multiplication or exponentiation in the appropriate finite field. S-boxes usually do not depend on the key. However, for some ciphers such as Blowfish [56], the s-boxes themselves depend on the key.

Consider a block cipher of R rounds and consider the reduced version of $R-1$ rounds. If, given a set of plaintexts, it is possible to distinguish whether some arbitrary data represents the ciphetext produced by encrypting these plaintexts by the $R-1$ rounds of the cipher for an unknown key, the R round cipher can be attacked as follows. For each guess for the R-th round subkey, the cryptanalyst checks whether the set of ciphertext can be distinguished as the output from the cipher on the given set of plaintext. If not, the last round subkey candidate is discarded. Since the last round subkey usually represents only a portion of the whole key, significant reduction of the keyspace is achieved. This attack

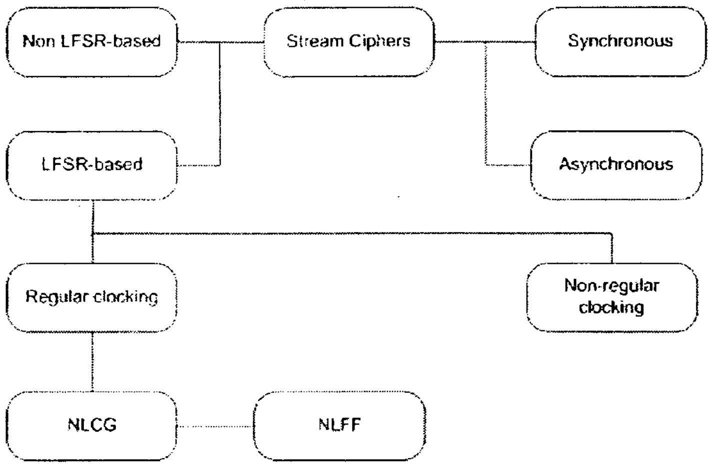

Figure 3: Classification of stream ciphers

method is used in several cryptanalytic attacks on block ciphers including linear cryptanalysis, differential cryptanalysis and interpolation attacks.

Let $m = (m_1, m_2, \ldots)$ denote the bit blocks of some fixed size w of the message to be encrypted. A stream cipher is a pair of efficient algorithms, one for transforming each of the messages m_i to ciphertext c_i and one for the inverse transformation. Both the encryption and decryption operations depend not only on the secret key K, but also on i which distinguishes stream ciphers from block ciphers. For example, if $m_l = m_d$ for some $l \neq d$, these two blocks will be transformed into two different ciphertexts when encrypted by a stream cipher. On the other hand, when encrypted by a block cipher, operating in the basic electronic code book (ECB) mode, these two blocks will be transformed into the same ciphertext. Frequent building-blocks for stream ciphers are keystream generators which can be thought of as finite state machines transforming the secret key K into a pseudorandom stream $z = (z_1, z_2, \ldots)$ where $z_1 \in \{0, 1\}^w$. In most of the cases, the z_i blocks are XOR-ed to the plaintext to produce the ciphertext stream.

As shown in Fig. 3, stream ciphers can be synchronous or asynchronous. For synchronous stream ciphers, the sender and receiver must maintain synchronization for the decryption operation to be successful. A stream of pseudo-random digits is generated independently of the plaintext and then applied to it for encryption. If synchronization is lost, various offsets need to be tried systematically to obtain correct decryption. In the case of asynchronous or self-synchronizing ciphers, previous N ciphertexts participate in computing the next keystream word. Thus, the receiver can automatically synchronize the keystream after receiving N ciphertexts. This makes it easier to recover if some ciphertext words are lost and consequently single-word errors are limited to N plaintext words.

Another classification of stream ciphers is with respect to their design structure. LFSR-based stream ciphers are those based on linear feedback shift registers, which are shift registers for which the feedback function is a linear function of its previous state. The initial value of the LFSR is called the *seed*. While sequences produced by LFSRs may have several interesting properties such as long period, balancedness and good correlation properties, the inherent linearity of these sequences limit their direct applications in cryptography. To ensure additional security properties, irregularly clocked LFSR based stream cipher design was proposed [21]. Such a design usually consists of more than one LFSR and at each step each LFSR is clocked. The choice of which of the output bits will be sent to the output of the cipher is a function of the outputs of LFSRs. A different way to destroy the nonlinearity of the produced sequence in a regularly clocked LFSR is to use more than one LFSR and use a non-linear combiner function (NLC generators). In order to ensure security properties of the produced stream, the function needs to satisfy several cryptographic properties such as balance, high nonlinearity, correlation immunity, high algebraic degree and high algebraic immunity degree. Another way to ensure desired properties of the output sequence is to use a non-linear filter function (NLFF) which operates on all bits of the LFSR.

Non-LFSR based stream ciphers include ciphers designed to be fast when executed in software. One of the most widely used ciphers in this category is RC4. Since RC4 operates with 8-bit words, it does not achieve the best possible performance on 32 bit and 64 bit processors which are in use today. Recent stream ciphers optimized for software usually operate with 32 and 64 bit words [4,13,23,50].

In the next chapter, we provide an overview of common cryptanalytic attacks on block ciphers and stream ciphers.

Chapter 2

Overview of cryptanalysis techniques

There exist several classifications of cryptanalytic attacks. First, we provide the one that differentiates attacks with respect to the information the cryptanalyst has access to:

- *Known ciphertext attack*: In this scenario, the cryptanalyst has access only to the ciphertext and not to the corresponding plaintext. The goal of the cryptanalyst in this case is to recover as many corresponding plaintexts as possible or to recover the key under which the encryption was done.

- *Known plaintext attack*: The cryptanalyst has both ciphertexts and their corresponding plaintexts. The cryptanalyst's goal is to recover the key under which the encryption was performed

- *Chosen plaintext attack*: The attacker has means to choose a set of plaintexts and obtain their corresponding ciphertexts. Again, the goal is to recover the secret key under which the encryption was performed.

Another classification of the attacks is with respect to whether the attacker has some sort of physical access to the encrypting device or not:

- *Pure mathematical attacks*: In this model, the attacker does not consider the physical implementation of the cipher and regards the problem as how to recover the secret key given plaintext/ciphertext information from a purely mathematical point of view.

- *Physical access dependant attacks*: In this model, the attacker has some physical access to the particular device that performs the encryption. This model includes side channel analysis in which the attacker measures certain parameters such as the instantaneous power consumption of the cryptographic device or the time used to perform the encryption operation. By utilizing this side channel information the attacker might be able to deduces some information about the internals of the encrypting process, which leads to key information recovery. Another cryptanalysis models that fall into this category is differential fault analysis of ciphers in which the attacker induces faults (errors) by applying physical influence such as ionizing radiation to the device during the encryption. By careful inspection of results of encryption in the faulty environment, the cryptanalysis might be able to gather information about the used secret key.

The success of a cryptanalytic attack in a general setting is measured by:

- *Amount of required input data*: The amount of ciphertext/plaintext information necessary to perform the attack.

- *Number of necessary operations*: The amount of necessary computations required to execute the attack. For example, in the case of a brute force attack in which every key is trivially checked, the number of operations is $2^{|K|-1}$ on average, where $|K|$ denotes the size of the key in bits.

- *Storage complexity*: The amount of storage required.
- *Number of necessary physical actions on the encrypting device*: This can include the number of necessary measurements in case of side channel analysis (such as power analysis attacks and timing attacks) or number of induced faults in the memory of the cipher, in case of fault analysis.

In the past twenty years a variety of efficient encryption algorithms have been developed. In parallel, the cryptanalysis community yielded many powerful attacks against ciphers. The most obvious attack on encryption algorithms is brute force attack, in which the cryptanalyst tries all the possible secret keys and finds the right one. As mentioned above, if the key is denoted by K, the number of necessary encryptions is around $2^{|K|-1}$. While the necessary plaintext-ciphertext pairs required for this attack is usually very small, most of modern ciphers are designed such that the key size is large enough to make this attack practically impossible.

In what follows we briefly review some of the most important attacks known today on both block ciphers and stream ciphers.

2.1 Block cipher cryptanalysis

In this section we give an overview of attacks on block ciphers.

Linear Cryptanalysis: In its basic version [41], linear cryptanalysis is a known plaintext attack that uses a linear relation, between some plaintext and ciphertext bits, that holds with probability different than $\frac{1}{2}$. The first part of linear cryptanalysis is to find linear approximations of the non-linear building blocks of the cipher, usually s-boxes. In other words, the relation among input and output bits of a given s-box of the form

$$X_{i_1} \oplus X_{i_2} \oplus \ldots \ominus X_{i_m} \oplus \ldots \oplus Y_{i_1} \oplus Y_{i_2} \oplus \ldots \oplus Y_{i_n}$$

needs to be established with the probability different than $\frac{1}{2}$, where X and Y denote the input and output of the s-box. For an s-box with n input bits and m output bits, there exist $(2^n - 1) \times (2^m - 1)$ possibilities and the cryptanalyst needs to investigate which ones hold with the high bias, defined as $\epsilon = p - \frac{1}{2}$. Next, these linear approximations are combined and a biased linear approximation between the key bits, the plaintext and ciphertext for the cipher reduced to $R - 1$ rounds is obtained. This is possible due to the fact that an XOR of several biased random variables is also a biased random variable, as given by the following Lemma which is usually referred to as the Piling-Up Lemma.

Lemma 1 *Let X_i be independent random variables of which the values are 0 with the probability of p_i and 1 with probability $1 - p_i$. Then, the probability that $X_1 \oplus \ldots \oplus X_n = 0$ is*

$$\frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} (p_i - \frac{1}{2})$$

Finally, the obtained $R - 1$ round distinguisher can be used to mount an attack by guessing the R-th round subkey bits and choosing the one for which most of the plaintext-ciphertext pairs satisfy the constructed linear relation. Linear cryptanalysis provided the most successful attack on DES, requiring 2^{43} known plaintext-ciphertext pairs.

Differential cryptanalysis: In its basic version [8], differential cryptanalysis studies how differences in the plaintexts affect the corresponding differences in the ciphertexts. As in linear cryptanalysis, the first part of differential analysis is to provide differential properties of the building blocks participating in the cipher. As for linear parts, such as bit permutations or key addition, the resulting differences are deterministically determined by the differences in the input. In case of non-linear components, such as s-boxes, knowledge of the input difference does not guarantee knowledge of the output difference. Instead, the cryptanalyst makes a table of all possible input/output differences, also called *differentials*:

$$\Delta X \xrightarrow{p} \Delta Y$$

where p denotes the probability that the input difference of ΔX will cause the output difference of ΔY. To obtain a differential for $R-1$ rounds of the cipher, differentials are combined along a *differential path* and the final differential probability is given by

$$\prod_{i=1}^{n} p_i$$

where n is the number of s-boxes used in the differential path and p_i the probability of the difference propagation within the i-th s-box. Again, the attack proceeds same as in linear cryptanalysis. The last round subkey is guessed and if the distribution of differences for the current key guess does not correspond to the expected one, the key candidate is discarded.

Since the last round subkey is usually smaller than the full key, significant reduction of the key space is achieved.

Generalizations of linear and differential cryptanalysis: Linear and differential cryptanalysis techniques have been generalized in different ways. In this part we provide an overview of these generalizations.

- *Differential-linear cryptanalysis [26]*: A chosen plaintext attack first applied to 8-round DES, in which differential cryptanalysis is applied to the first three rounds and linear cryptanalysis is applied on the remaining five rounds. The main observation on which the attack relies is that inverting certain bits in the input of the first round leaves certain third round bits unchanged, implying that the XOR sum of these third round bits is also left unchanged. From rounds four to seven, a linear approximation involving exactly these unchanged four-round input bits and certain 7-th round bits is then used. For each 8-th round subkey portion, it is verified whether for each plaintext, the XOR sum of the bits in questions changes or not. Using differential-linear cryptanalysis, the number of necessary chosen plaintext-ciphertext pairs was reduced to 512 to recover 10 bits of the key, for 8-round DES. The classical Biham-Shamir differential attack required over 5000 chosen pairs.

- *Truncated differential cryptanalysis [30]*: In a conventional differential attack, all the bits in the input and output difference are specified and the differential is denoted by (a, b) where a and b are of the same length as the plaintext for the cipher. In truncated differential cryptanalysis, not the whole differences are specified. In [30], truncated differentials have been used to attack a 6-round DES with only 46 chosen plaintext-ciphertext pairs. Also, in [31], it has been shown that there exists a 24-round Skipjack truncated differential that holds with probability 1.

- *Higher-order differential cryptanalysis [32]*: Whereas ordinary differential cryptanalysis considers differences between plaintexts defined as

$$\Delta_a f(x) = f(x \oplus a) \oplus f(x)$$

higher-order differential cryptanalysis utilizes the i-th derivative of the function f

$$\Delta^i_{a_1,\dots a_i} f(x) = \Delta_{a_i}(\Delta^{(i-1)}_{a_1,\dots a_{i-1}} f(x))$$

It has been shown that it is possible to construct a cipher that is unbreakable by means of classical differential cryptanalysis and at the same time weak with respect to higher order differential cryptanalysis, making the attack relevant. For instance, in [32] it was shown that for the function $f(x, k) = (x + k)^2 \bmod p$ with input/output size of $2 \times log_2 p$, where p is prime, every non-trivial one round differential has probability of $\frac{1}{p}$ and the second order derivative is a constant. The problem with high-order differentials is to combine them to more than two rounds, as it is possible with first order differentials.

- *Impossible differential cryptanalysis [6]*: Instead of using biased differentials, differentials with probability 0 at some intermediate state of the cipher are used to derive key information. The last-round keys are discarded on the basis of the indication that the impossible event happened during the encryption if this key part was used. The attack was applied to Skipjack reduced to 31 rounds.

- *Boomerang attack [64]*: In its basic version, the attack requires chosen-plaintext and chosen-ciphertext queries to combine two high-probability differential paths which are not necessarily related to each other. To show that differential cryptanalysis of ciphers is infeasible, cipher designers usually compute an upper bound p on the probability of any differential characteristics of the cipher and then apply the often repeated "folk theorem" to show that the differential attack would require at least $\frac{1}{p}$ texts to break the cipher. According to [64], this folk theorem is wrong, in case that the attacker can adaptively choose ciphertext for which the plaintext will be obtained.

Other attacks, independent of linear and differential cryptanalysis include:

- *Interpolation attacks [27]*: This attack can be applied to ciphers for which the round function can be written as an algebraic expression with low degree or sparse. For ciphers in which s-boxes are used, the first step is to use the following theorem to find the coefficient of the polynomial that will correspond to the s-box:

Theorem 1 *Let K be a field. Given $2n$ elements $x_1, \ldots x_n, y_1, \ldots y_n \in R$, where x_i's are distinct, define:*

$$f(x) = \sum_{i=1}^{n} y_i \prod_{1 \leq j \leq n, j \neq i} \frac{x - x_j}{x_i - x_j}.$$

Then $f(x)$ is the only polynomial over K of degree at most $n-1$ such that $f(x_i) = y_i$, for $i = 1, \dots n$.

The expression for f is called the *Lagrange interpolation formula*. Next, the round polynomial functions are combined to represent the whole cipher as a polynomial, in which the coefficients are key dependent. By using a sufficient number of chosen plaintext-ciphertext pairs, all coefficients may be computed and the cipher will be considered broken, even without computing the key bits. This is why the attack in its basic form is called a *global deduction* attack. In other words, an alternative representation of the cipher is found, but the key is not necessarily obtained. However, the last-round key guessing technique allows it to be converted to a key recovery attack.

- *Square attacks, Integral attacks, Multiset attacks [12,16]*: Unlike in, say linear cryptanalysis, in which every plaintext-ciphertext pair can bring the cryptanalysis a piece of information, in multiset cryptanalysis, information can be obtained only by considering the whole set of plaintext-ciphertext pairs. The attack relies on the properties of building blocks which conserve certain properties of the set of input values. Among such properties are for example the distinctiveness of elements in the set or their sum. Consider for example the effect of applying a bijective s-box to each element of a multiset in which every possible plaintext appears the same number of times. For instance, if there is $k \times 2^n$ plaintexts in the set, every value appears k times. Trivially, the property will be preserved in the set of corresponding ciphertexts as well. Multiset attacks trace this kind of properties through as many rounds as possible. Integral cryptanalysis refers to the attack when the used set property is that $\sum_{x \in G} x = 0$, where G is the group that the elements belong to, i.e., the set is *balanced*. The property is conserved after addition of two sets and this is what is important for pushing the property through several rounds in the cipher. This attack was first used to cryptanalyze the cipher Square [16].

- *Related key attacks [7]*: This type of attack refers more to another cryptanalysis model than to a specific attack method on ciphers. Namely, in related-key cryptanlaysis, the cryptanalyst is able to learn ciphertexts of some plaintexts not only under the original unknown key, but also under a key in some preset relation with the original key. In general, the attack can be mounted if a relation between keys K and K' and a relation between P and P' results in some relation between corresponding ciphertexts C and C'.

  While it might appear that the interest of related-key attacks is purely theoretical, examples of communication protocols exist where simple key management makes relate-key cryptanalysis practical. The security of the cipher when used in hashing mode can also be affected by this type of weaknesses.

- *Slide attacks [11]*: The specific property of this attack is that it usually does not depend on the number of rounds used in the cipher. The attack relies on the key schedule weakness in which parts of the key are reused subsequently in different rounds. Assume for instance, that the same key is used in each round transformation and this property of the cipher be called *self-similarity*. The attack starts by finding *slid pairs* which are defined as follows.

Definition 1 *Let F be the round function of an iterated block cipher. If a pair of known plaintexts $(P, C), (P', C')$ satisfies $F(P) = P'$, then due to the self-similarity of both the rounds and the key schedule, the corresponding ciphertexts also satisfy $F(C) = C'$. Such a pair is called a slid pair.*

By finding a slid pair, which is possible using $O(2^{\frac{n}{2}})$ plaintexts due to the birthday paradox, the attacker obtains a plaintext P and its one-round ciphertext, P'. If from pair (P, P') it is possible to deduce the information about key K, the secret key is compromised.

2.2 Stream cipher cryptanalysis

In this section we provide an overview of attacks on stream ciphers.

Berlekamp-Massey Algorithm [2]: In 1967, Berlekamp presented an algorithm for decoding certain type of codes, called Bose-Chaudri-Hocquenghem codes [2]. Two years after, Massey successfully applied this algorithm to LSFRs. In that form, the algorithm finds the shortest linear feedback shift register that produces a given sequence of bits. A generalization of this algorithm is Reeds-Sloane algorithm that finds the shortest LFSR for a given output sequence, when the elements in the sequence take values from integers mod n. The algorithm is used for cryptanalysis of LFSR designed stream ciphers if the linear complexity of the resulting key stream is not long enough.

Correlation attacks [58]: This attack is applicable to ciphers that use a Boolean function to combine several linear feedback shift registers to produce the keystream output. If a Boolean function is poorly chosen, only then the correlation attack applies. By careful choice of the combining Boolean function, this attack can be mitigated and therefore the correlation weaknesses are not inherent to the design itself. The attack works as follows. Suppose that the keystream is produced by combining $x_0, \dots x_8$, each produced from a different LFSR, by a Boolean function f. Assume also that there exists significant correlation between x_0 and $f(x_0, \dots x_8)$, i.e., that for example $x0 = f(x_0, \dots x_8)$ in 75% of cases. To recover the internal state of the $LFSR$ corresponding to x_0, all that is needed is to go through all the possibilities of this LFSR and see which one has around 75% same bits as the keystream output sequence. The Geffe generator [20] is a well known example for stream ciphers broken by this technique.

Guess and Determine Attacks: The idea in this type of attack is to guess certain parts of the internal state and then combine this with certain keystream output words to deduce more bits of the internal state. If an inconsistency with other keystream words is observed, the guess for the internal state is discarded. A wide variety of stream ciphers have been tested with respect to this type of the attack, from RC4 to eSTREAM candidates such as SOSEMANUK. In the attack on RC4 [33], whereas the effective size of the internal state is approximately 1800 bits, the guess and determine attack reduces it to around 700 bits. Guess and determine analysis of SOSEMANUK [1] shows that the internal state of size 384 bits, can be recovered using only 2^4 keystream words and requires 2^{224} operations. It can be concluded that the guess and determine attacks are a powerful technique for finding the internal state of the cipher and that special care must be taken during the design of the cipher for mitigating this type of weaknesses.

Key scheduling weaknesses: A breakthrough in RC4 cryptanalysis was achieved by Mantin's observation [36] that the second RC4 keystream byte produced by different randomly distributed keys is biased. The observation lead to possible recovery of the second plaintext byte if RC4 was used in broadcast mode in which the same plaintext was encrypted by several hundreds of unrelated different keys. The attack put into focus the need for the internal state to be properly randomized after the key scheduling phase and subsequently RC4 [5] and other stream ciphers were systematically analyzed with respect to this property.

Throughout the rest of this thesis, we present some of our cryptanalytic results on some recently proposed block and stream ciphers.

Chapter 3

Cryptanalysis of two light-weight block ciphers

In this chapter we provide attacks on TREYFER [70] and PIFEA-M [15] block ciphers. TREYFER is a cipher designed for resource constrained environments. We provide a related-key attack on TREYFER which recovers the secret key with relatively low data and computational complexity. PIFEA-M is an improvement of IFEA-M, Improved Fast Encryption Algorithm for Multimedia [46], aimed for data encryption of multimedia data and designed to resist certain implementation dependent attack on IFEA-M. We show that PIFEA-M is not resilient to a similar style attack.

3.1 A Related-Key Attack on TREYFER

Despite its current implementation advances, the Advanced Encryption Standard (AES) [49] may not always be the optimal choice for applications with tight resource constraints such as radio frequency identification (RFID) tags and tiny sensor networks. In fact, with the widespread applications of these resource-constrained devices, the analysis and design of lightweight encryption algorithms have started to gain a new momentum (e.g [14], [63].)

TREYFER [70] is a 64 bit block cipher (and also a MAC) proposed by Gideon Yuval, from Microsoft, at FSE'97. According to Gideon Yuval, TREYFER is targeting an environment for which even TEA [65] and SAFER [39] are "*gross overdesign* [70]". The simple and compact design of TREYFER makes it an attractive choice for resource constrained environments such as smart cards, RFIDs, and sensor networks. For example, TREYFER requires only 29 bytes of executable code on the 8051 micro-controller.

The best known attack against TREYFER, presented by Alex Biryukov and David Wagner [11], is a slide attack that requires 2^{32} known plaintexts, 2^{44} time for analysis and 2^{32} memory.

In this chapter, we derive a set of deterministic algebraic relationships between the ciphertexts corresponding to related plaintexts encrypted with TREYFER under circularly byte shifted versions of the same key. Based on these relationships, we present a chosen related-key attack [7]- [10] that directly recovers the secret key of TREYFER using about 2^{11} chosen plaintext encryption operations. The attack complexity is independent of the number of rounds of the cipher.

3.1.1 Description of TREYFER

TREYFER can be seen as an iterative block cipher with the round function shown in Figure 4 where $<<<$ denotes a circular left shift by 1 bit, $+$ denotes addition mod 256 and SKi, $i = 0 \ldots 7$, denotes the key addition and s-box lookup operations $S[(x + K_i) \bmod 256]$ function. All operations are byte-oriented, and there is a single 8×8-bit s-box.

In [70], the s-box is left undefined; it is suggested that the implementation can simply use whatever data is available in memory. In each round, each byte has added to it the s-box value of the sum of a key byte and the previous data byte, then it is rotated left one bit. The design attempts to compensate for the simplicity of this round transformation by using a large number of rounds: 32.

The pseudo code implementation of TREYFER is as follows:

```
$for(r = 0; r < NumRounds; r++)\{$
$text[8] = text[0];$
$for(i = 0; i < 8; i++)$
$text[i+1] =$
$(text[i+1] + S[(key[i] + text[i])\%256]) <<< 1;$
$text[0] = text[8];$
$\}$
```

In order to obtain a more compact and faster cipher under the assumed hardware constraints, the designer of TREYFER opted not to have any complex key scheduling. In particular, TREYFER simply uses its user supplied key, K, byte by byte in exactly the same way at each round.

The following notation will be used throughout the this part of the chapter.

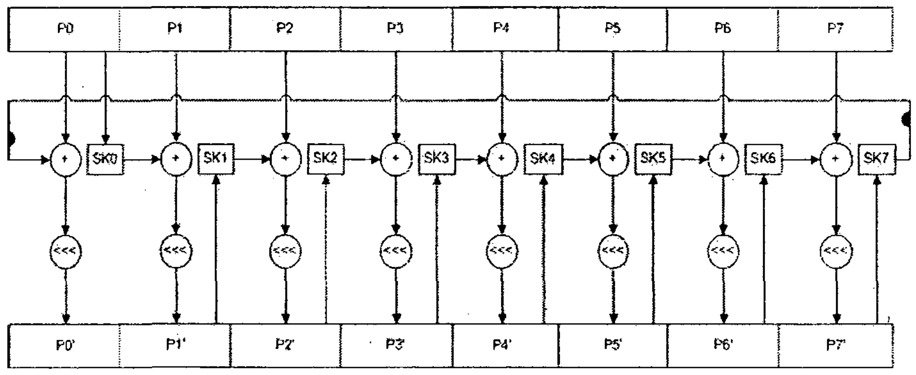

Figure 4: Round function of TREYFER, where $SKi(x) = sbox((x + K_i) \, mod \, 256)$

- f denotes round function mapping of TREYFER (see Figure 1).
- $P = P_0P_1 \cdots P_7$ denotes the 8 byte plaintext input
- $K = K_0K_1 \cdots K_7$ denotes the 8 byte key
- $C = C_0C_1 \cdots C_7$ denotes the 8 byte ciphertext

3.1.2 Main Observations

In this subsection, we derive an algebraic relationship between the ciphertexts corresponding to related plaintexts encrypted under circularly (byte-wise) shifted versions of the secret key.

Lemma 2 *Let*

$$P_0'P_1' \cdots P_7' = f(P_0 \cdots P_7, K_0 \cdots K_7)$$

and

$$P_0''P_1'' \cdots P_7'' = f^2(P_0 \cdots P_7, K_0 \cdots K_7).$$

Then we have

$$f(P_1'P_2\cdots P_7P_0, K_1\cdots K_7K_0) = P_1''P_2'\cdots P_7'P_0'$$

Proof: Let $p_i, i = 0\ldots7$ denote i-th byte of $f(P_1'P_2\cdots P_7P_0, K_1\cdots K_7K_0)$. Then, by TREYFER definition, we have:

$$\begin{aligned}
p_1 &= (P_2 + S(P_1', K_1)) <<< 1 = P_2' \\
p_2 &= (P_3 + S(P_2', K_2)) <<< 1 = P_3' \\
&\vdots \qquad\qquad\qquad\qquad \vdots \\
p_7 &= (P_0 + S(P_7', K_7)) <<< 1 = P_0' \\
p_0 &= (P_1' + S(P_0', K_0)) <<< 1 = P_1''
\end{aligned}$$

The lemma holds by noting that $f(P_1'P_2\cdots P_7P_0; K_1\cdots K_7K_0) = p_0\cdots p_7$.

This observation can easily be extended to hold for composition of multiple rounds. By $P \xmapsto{K} C$ we will denote the TRYEFER encryption of the plaintext P with a key, K.

Lemma 3 *Let*

$$P_0\cdots P_7 \xmapsto{\mathrm{K}} C_0\cdots C_7,$$

and

$$C_0'\cdots C_7' = f(C_0\cdots C_7, K_0\cdots K_7).$$

Then

$$P_1'P_2\cdots P_7P_0 \overset{\mathrm{rot(K,1)}}{\longmapsto} C_1'C_2\cdots C_7C_0$$

Proof: Follows from previous Lemma and the fact that TREYFER function is equal to f^n, $n = 32$.

That way, for each TREYFER pair (P, C), we can derive another "similar" plaintext-ciphertext pair, encrypted by key circularly shifted to the left by one byte. Furthermore, if previous Lemma is applied multiple times, we can get 7 such pairs, as given by the following Theorem.

Theorem 2 *Let* $\mathrm{rot}(K, i)$ *denote the left circular shift of* K *by* i *bytes, then, for any*

$$P_0 \cdots P_7 \overset{K}{\longmapsto} C_0 \cdots C_7 \tag{1}$$

we have

$$P_1' P_2 P_3 P_4 P_5 P_6 P_7 P_0 \overset{\mathrm{rot}(K,1)}{\longmapsto} C_1' C_2 C_3 C_4 C_5 C_6 C_7 C_0 \tag{2}$$

$$P_2' P_3 P_4 P_5 P_6 P_7 P_0 P_1' \overset{\mathrm{rot}(K,2)}{\longmapsto} C_2' C_3 C_4 C_5 C_6 C_7 C_0 C_1' \tag{3}$$

$$\vdots \qquad\qquad \vdots$$

$$P_7' P_0 P_1' P_2' P_3' P_4' P_5' P_6' \overset{\mathrm{rot}(K,7)}{\longmapsto} C_7' C_0 C_1' C_2' C_3' C_4' C_5' C_6' \tag{8}$$

It should be noted that the above presented property of TREYFER does not depend on any particular choice of the s-box.

3.1.3 The Attack

Related-key cryptanalysis assumes that the attacker learns the encryption of certain plaintexts not only under the original unknown key, K, but also under some related keys (e.g., $K' = g(K)$). In a chosen-related-key attack, the attacker specifies how the key is to be changed. It should be noted that the attacker knows or chooses the relationship between keys, i.e., $g(\cdot)$, but not the actual key values.

Based on the relations derived in the above subsection, we describe a chosen-related-key attack against TREYFER.

Given the plaintext-ciphertext pair (P, C), $P \xmapsto{K} C$ where $K = K_0 \cdots K_7$, the proposed attack proceeds to recover K_0 as follows:

```
For( $X = 0; X < 256; X++$) {
    • Encrypt the plaintext $XP_2 \cdots P_7P_0$ under the key $\mathrm{rot}(K, 1) = K_1 \cdots K_7K_0$
    • For each ciphertext in the form $YC_2 \cdots C_7C_0$, determine $K_0$ that satisfies
          $X = P_1' = (P_1 + S[K_0 + P_0]) <<< 1,$
          $Y = C_1' = (C_1 + S[K_0 + C_0]) <<< 1.$
}
```

Figure 5: TREYFER key recovery algorithm

The process above finds P_1' and C_1', the second bytes of $f(P_0 \cdots P_7, K_0 \cdots K_7)$, and $f(C_0 \cdots C_7, K_0 \cdots K_7)$, respectively.

Theoretically it is possible that there may exist an $X \neq P_1'$ such that $XP_2 \cdots P_7P_0 \xmapsto{\mathrm{rot(K)}} YC_2 \cdots C_7C_0$, leading to false information on K_0. However, the probability that this will happen is practically negligible ($\approx 3 \times 10^{-15}$).

If the s-box is bijective, then only one of the equations above can be used to uniquely determine $K_0 = S^{-1}[(P_1' >>> 1) - P_1] - P_0$. In the case of non bijective s-boxes, the above two steps can be repeated until K_0 is uniquely determined.

K_1 to K_6 can be sequentially recovered by performing the above steps using related plaintexts and keys, according to relations (3)-(8). Similarly, the last byte of the key, K_7, can be recovered using relation (1) or simply by exhaustive search.

Thus, the attack requires about $8 \times 256 = 2^{11}$ chosen plaintext-ciphertext pairs, each, 256 of them are encrypted under a key that is circularly bytes shifted version of the original secret key.

Remark 1 *The above attack can be thought of as a slide attack in which the sliding pairs are produced at the s-box level and not at the level of the whole round function.*

3.2 Cryptanalysis of PIFEA-M

Confidentiality plays one of the key roles in proper implementation of multimedia applications over the Internet. Due to the widespread existence of eavesdropping and hacking tools, privacy of the content has to be ensured for users exchanging multimedia data such as video or voice. Encryption is one of the basic tools to achieve this privacy. The volume of information exchanged in multimedia applications is high and encryption algorithms designed for this purpose have to be fast. At the same time, the algorithm has to be secure enough.

To address this problem, Yi *et al.* [67] proposed Fast Encryption Algorithm for Multimedia (FEA-M). The design of the cipher utilizes Boolean matrices and does not follow a typical scheme of neither block nor stream cipher design. In [69], a practical adaptive chosen plaintext attack on FEA-M, requiring only 1.5 kilobytes of data, has been presented. Similar attack was also described in [45]. In [46], it is shown that the underlying system of FEA-M nonlinear equations can be solved in a much more efficient way than in general case. In the same paper, IFEA-M cipher has been proposed by modifying FEA-M so that it resists algebraic attacks and to provide tolerance to packet loss errors.

A problem with possible improper implementation of IFEA-M has been pointed out in [34]. Assuming that the attacker is able to force the user to use the same session key twice (for example by controlling the pseudorandom generator through public time service), the attacker would then be able to find the master key using a differential known plaintext attack. To defend against this differential attack, Chefranov [15] proposed PIFEA-M, a parameterized version of IFEA-M, and estimated that its performance is around 25% better than of IFEA-M. As for resistance to previously reported software dependent differential attack, in [15] it is claimed that breaking the cipher requires at least $O(2^{72})$ operations, which is practically infeasible using current technologies.

In this chapter we show that, under the same assumptions, PIFEA-M is still vulnerable to differential attacks. Let (P_i^j, C_i^j) denote a plaintext-ciphertext pair produced by i-th encryption in session j. Given the pairs (P_1^1, C_1^1), (P_1^2, C_1^2), (P_2^1, C_2^1) and (P_2^2, C_2^2), we show that the attacker can recover all other successive plaintexts P_i, $i \geq 3$ from both sessions with very small computational complexity.

3.2.1 PIFEA-M specification

In this subsection, we briefly review the specifications of PIFEA-M. For further details, the reader is referred to [15]. PIFEA-M encrypts $n \times n$ Boolean matrices using $n \times n$ key. Master key K_0 is assumed to be shared by the users in advance and the steps to achieve common secret matrix are described in [68]. Session key K, initial matrix V and parameter matrix R are generated by the sender and transmitted to receiver as follows:

$$\begin{aligned} K^* &= K_0 \cdot K^{-1} \cdot K_0 \\ V^* &= K_0 \cdot V \cdot K_0 \\ R^* &= K_0 \cdot R \cdot K_0 \end{aligned} \tag{9}$$

The receiver discloses obtained data as follows:

$$\begin{aligned} K^{-1} &= K_0^{-1} \cdot K^* \cdot K_0^{-1} \\ V &= K_0^{-1} \cdot V^* \cdot K_0^{-1} \\ R &= K_0^{-1} \cdot R^* \cdot K_0^{-1} \end{aligned} \tag{10}$$

The parameter matrix R contains five n-bit numbers r_k, $k = 1, \ldots 5$ contained by the first five rows of the matrix. All other elements of the matrix are set to zero. Session K and initial matrix V are generated randomly by sender.

The message to be transmitted is padded with zeros if needed and then divided to blocks $P_1, \ldots P_r$ of size n^2. These blocks are then arranged as matrices of dimension $n \times n$ and encrypted and decrypted as follows:

$$\begin{aligned} C_i &= (P_i \oplus A_{r_1 r_2}) B_{r_3 r_4 r_5, i} \oplus A_{r_1 r_2} \\ P_i &= (C_i \oplus A_{r_1 r_2}) B^{-1}_{r_3 r_4 r_5, i} \oplus A_{r_1 r_2}; \end{aligned} \tag{11}$$

where $A_{r_1 r_2} = V^{r_1} K^{r_2}$, and $B_{r_3 r_4 r_5, i} = V^{r_3} K^{r_4 + i} V^{r_5}$.

3.2.2 The attack

We present a differential style known plaintext attack, provided that the following assumption holds:

Assumption 1 *It is assumed that the same session key matrices K and V are used in two sessions [15].*

Again, we stress the fact that the cipher designer in [15] explicitly specified that the cipher is secure under the above assumption. In fact, resisting such attacks was the main design goal of PIFEA-M.

Given the four plaintext-ciphertext pairs (P_i^1, C_i^1), (P_i^2, C_i^2), (P_{i+1}^1, C_{i+1}^1), (P_{i+1}^2, C_{i+1}^2), our proposed attack proceeds as follows:

1. Considering the difference between C_i^1 and C_i^2 yields:

$$\begin{aligned} \Delta C_i &= ((P_i^1 \oplus A_{r_1, r_2}) \cdot B_{r_3, r_4, r_5, i} \oplus A_{r_1, r_2}) \\ &\oplus ((P_i^2 \oplus A_{r_1, r_2}) \cdot B_{r_3, r_4, r_5, i} \oplus A_{r_1, r_2}) \\ &= \Delta P_i \cdot B_{r_3 r_4 r_5, i} \end{aligned} \tag{12}$$

Substituting $i = 1$ and $i = 2$ in (12) and using the four plaintext-ciphertext pairs, the values of $B_{r_3r_4r_5,1}$ and $B_{r_3r_4r_5,2}$ can be obtained.

2. From the cipher specification, we have:

$$\begin{aligned} B_{r_3r_4r_5,1} &= V^{r_3} \cdot K^{r_4+1} \cdot V^{r_5} \\ B_{r_3r_4r_5,2} &= V^{r_3} \cdot K^{r_4+2} \cdot V^{r_5} \end{aligned} \tag{13}$$

Inverting both sides of the first equation and then multiplying the equations gives

$$B_{r_3r_4r_5,2}B^{-1}_{r_3r_4r_5,1} = V^{r_3}KV^{-r_3} \tag{14}$$

By noting that

$$\begin{aligned} &B_{r_3r_4r_5,i+1}B^{-1}_{r_3r_4r_5,i} &= \\ &V^{r_3}K^{r_4+i+1}V^{r_5}(V^{r_3}K^{r_4+i}V^{r_5})^{-1} &= \\ &V^{r_3}KV^{-r_3}, \end{aligned} \tag{15}$$

then, from (14), we get:

$$B_{r_3r_4r_5,i+1} = B_{r_3r_4r_5,2} \cdot B^{-1}_{r_3r_4r_5,1} \cdot B_{r_3r_4r_5,i} \tag{16}$$

Thus $B_{r_3r_4r_5,i}$ for $i \geq 3$ can be calculated recursively.

3. As for $A_{r_1r_2}$, it can be easily derived by solving the linear matrix equation (for $i = 1$ or $i = 2$):

$$A_{r_1r_2} = (C_i \oplus P_iB_{r_3r_4r_5,i})(B_{r_3r_4r_5,i} \oplus I)^{-1}, \tag{17}$$

where I denotes the identity matrix.

Finally, the knowledge of $B_{r_3r_4r_5,i}$ and $A_{r_1r_2}$ enables the attacker to decrypt the plaintext corresponding to any ciphertext C_i, $i \geq 3$ in the assumed two sessions.

Chapter 4

A new distinguishing and key recovery attack on NGG stream cipher

NGG is an RC4-like stream cipher designed to make use of today's common 32-bit processors. It is 3-5 faster than RC4. In this chapter, we show that the NGG stream can be distinguished, with success probability $\approx 97\%$, from a random stream using only the first keystream word. We also show that the first few kilobytes of the keystream may leak information about the secret key which allows the cryptanalyst to recover the secret key in a very efficient way.

4.1 Introduction

Because of its simplicity and speed, RC4 [44] is one of the most widely used stream ciphers in software applications. It is implemented in many protocols and applications such as

Secure Socket Layer (SSL), and Wired Equivalent Privacy (WEP).

Typically, RC4 operates with 8-bit values both on output and in the internal state. Ciphertext is obtained by XOR-ing keystream bytes with the plaintext. From the perspective of modern processors with 32/64-bit word size, this is inefficient. A stream cipher that produces 32/64-bit keystream words and requires similar number of operations per step would be around 4-8 times faster, since an 8-bit operation on these processors takes equal time as a 32/64-bit operation. To address this problem, several generalizations of RC4-like stream ciphers have been proposed (e.g., RC4A [52], VMPC [72], NGG [50] and GGHN [23]).

The NGG cipher was proposed by Nawaz *et al.* [50] [61]. Originally, NGG was named RC4(n,m) where m denotes the bit-length of the keystream output word and the size of the internal state table is 2^n. Later on, the name NGG was adopted for this cipher. Another version of this cipher, called GGHN, was introduced in [23], but our focus of this chapter is the original version of the cipher.

In this chapter, we show that key schedule algorithm (KSA) of NGG is flawed. This allows the cryptanalyst to distinguish NGG from a random stream using only the first keystream word. Furthermore, we show that the resulting statistical bias in the key scheduling process allows the cryptanalyst to recover information about the secret key from the first few kilobytes of the keystream output.

The last decade has witnessed an extensive cryptanalytic literature on RC4 including distinguishing attacks (e.g., [18, 22, 37]), internal state recovery attacks (e.g., [33, 37, 43, 60]), and attacks on the key scheduling algorithm (e.g., [5, 19, 36, 38, 47, 53, 54, 59, 62]). Recently, analyzing the security of generalized RC4-like ciphers has also gained some momentum (e.g., [42], [51], [61], [66]). The cryptanalytic results presented in this chapter resembles the work in [5, 53], in which it is shown that internal state permutation table of RC4, right after KSA, can leak secret key bytes. On the other hand, apart from [29] which studies weaknesses associated with concatenating IVs to the key, this is the first time that the security of NGG key schedule algorithm is addressed. Also, according to our knowledge, no key recovery algorithms on NGG have been proposed until now. Best previously published NGG distinguisher [66] exploited a problem in NGG pseudorandom number generation algorithm and requires around 100 consecutive keystream words. The distinguisher presented in this chapter requires only the first key stream word generated right after the the KSA. The attack presented in [61] focuses on GGHN and uses the first two keystream words associated with about 2^{30} secret keys to build a distinguisher. While this attack may also be applicable to NGG, the high frequency of key changing required by this attack (2^{30} keys) questions its practical significance against both ciphers.

Among cryptographers, there exist different stances in the debate on whether distinguishing attacks represent a threat to the security of stream ciphers or not. For example, Rose [25] argues that, unlike the block ciphers case, most of the distinguishing attacks on stream ciphers do not represent a real threat to the practical security of the cipher and hence one should make a distinction between powerful distinguishing attacks and weak ones based on whether the attack may lead to deriving useful cryptanalytic information such as key bits or not. Conversely, according to Bernstein [3], even distinguishing attacks that do not yield key or other cryptanalytic information are more than mere certificational weaknesses. An attacker, at least for some plaintext distributions, might be able to detect change of entropy in the plaintext only by looking at the ciphertext. If for example, a dummy message with high entropy is sent among sender and receiver from time to time to foil the attacker's analysis, the attacker might be able to use a distinguisher to isolate and discard these dummy messages.

The statistical bias weakness presented in this chapter is such that it allows distinguishing the cipher from a random stream and, at the same time, recovering some secret key information. It should be noted that the exhibited weakness is in key scheduling phase of the algorithm and not in keystream generation part of the algorithm. In particular, the NGG keystream generation procedure creates biased internal state and this is detectable in the first few kilobytes of the cipher. Thus, in a way, by observing this bias, it is natural to expect that information about the secret key might be revealed.

The rest of the chapter is organized as follows. In section 2, the relevant details of the NGG cipher are given and previous attacks are described. Non-randomness of the S table after the NGG KSA is proved in section 3. In section 4, a distinguisher utilizing this weakness is constructed, and the success probability estimates are given. In section 5, we show how it is possible to recover information about the secret key by looking at the first few kilobytes of the keystream output. We conclude in section 6.

4.2 The NGG stream cipher

NGG(n, m) denotes a parameterized family of ciphers. For some fixed n and m, the NGG internal state consists of a public n-bit counter i, secret pseudorandom n-bit counter j and

S table consisting of $N = 2^n$ m-bit values. The cipher consists of two separate algorithms (see Figure 6 where $M = 2^m$):

- Key Scheduling Algorithm (KSA): In the *initialization* step of the KSA, the table S is initialized with prespecified publicly known random array **a**. Then, in the *scrambling* step of the KSA, these values are mixed depending on key bytes in a pseudorandom value as follows: the i-th and the j-th value are swapped and the sum of these two values is assigned to the i-th element. This is repeated for $i = 0, \ldots N - 1$, where j is incremented pseudorandomly depending on the key.

- Pseudo Random Number Generation (PRNG): First, counters i and j are updated. Then, $S[i]$ and $S[j]$ values are swapped. Value $S[(S[i] + S[j]) \bmod N]$ is sent to the output and then changed to $S[i] + S[j]$.

Before using the cipher, it needs to be initialized by the KSA supplied with the secret key. The output of this process is a randomized secret internal state of NGG. During the encryption process, m-bit plaintext words are XORed with m-bit keystream words produced by the PRGA. Similarly, during the decryption process, the m-bit ciphertext words are XORed with the corresponding keystream words.

In [66] it was shown that NGG is distinguishable from a random sequence with about 100 keystream words. The attack relies on the fact that for three random S table entries, the relation $S[X] = S[Y] + S[Z]$ holds with biased probability, due to the update step of NGG. In [51], it was found that the least significant bit of NGG keystream word is biased, due to "bias inducing state", which occurs with probability 2^{-16}. Distinguisher based on this can

```
KSA
Initialization:
For $i = 0, ..N-1$
  $S[i] = a_i$
$j = 0$
Scrambling:
For $i = 0, ..N-1$
  $j = (j + S[i] + K[i \bmod l]) \bmod N$
  Swap($S[i], S[j]$)
  $S[i] = (S[i] + S[j]) \bmod M$
```

```
PRGA
Initialization:
$i = j = 0$
Loop:
  $i = (i+1) \bmod N$
  $j = (j + S[i]) \bmod N$
  Swap($S[i], S[j]$)
  $t = (S[i] + S[j] \bmod M) \bmod N$
  Output=$S[t]$
  $S[t] = (S[i] + S[j]) \bmod M$
```

Figure 6: NGG(n, m) specification

be built by using $2^{32.89}$ keystream words. In Klein's work [29], in which RC4 used in WEP mode is shown to be weak, NGG is shown to be prone to a similar attack, which makes it insecure when IVs are concatenated to keys [29].

Unlike previous distinguishing attacks, our attack can naturally be extended to a key recovery attack. Throughout the rest of this chapter, we consider the popular case of $n = 8$ and $m = 32$. It should be noted, however, that the attacks described in this chapter becomes more sever as m increases (e.g., for $m = 64$).

4.3 Weakness in NGG KSA

In the initialization step of NGG KSA, $S[k]$ element is assigned $\mathbf{a}_k$, $k = 0 \ldots 255$, where array **a** is publicly known. One choice for this array is given in [50]. Then, in the scrambling step, for $i = 0, 1, 2, \ldots 255$, the i-th value is swapped with pseudorandom element of S and place i is assigned the sum of these two elements.

In this section, we show that the scrambling step does not randomize the S table sufficiently. Namely, after the KSA, most of the elements of S can be represented as a sum of one, two, three or four elements of **a** values. In other words, there is a high probability that, for random index k, one of the following four relations will hold:

- $S[k] = \mathbf{a}_{x_1}$ for some index x_1,
- $S[k] = (\mathbf{a}_{x_1} + \mathbf{a}_{x_2})$ mod M, for some indices x_1, x_2,
- $S[k] = (\mathbf{a}_{x_1} + \mathbf{a}_{x_2} + \mathbf{a}_{x_3})$ mod M, for some indices x_1, x_2, x_3,
- $S[k] = (\mathbf{a}_{x_1} + \mathbf{a}_{x_2} + \mathbf{a}_{x_3} + \mathbf{a}_{x_4})$ mod M, for some indices x_1, x_2, x_3, x_4,

where $0 \leq x_i \leq 255$. As will be shown, this is highly improbable for a randomly chosen 32-bit word. In the following we prove this observation by modelling the KSA procedure.

4.3.1 Definitions and assumptions

We say that the KSA is at step i, $i = 0 \ldots 256$, if i scrambling steps were executed. For example, KSA is at step 0 right after initialization step of KSA, and before the first scrambling step and at step 256 after it is finished. Accordingly, by j_i and S_i we denote j and S at step i of the KSA.

Let $W_n = \{\sum_{i=1}^{n} a_{x_i} | \; x_i \in \{0..255\}\}$. Unless otherwise specified, we always use the set of **a** values proposed in [50]. It should be noted, however, that changing the set **a** cannot prevent the attacks described in this chapter. In general, $W_i, i = 1, 2, \cdots$ sets are not disjoint. However, for the choice of **a** in [50], $W_1 \cap W_2 = \emptyset$, $W_1 \cap W_3 = \emptyset$ and $W_2 \cap W_3 = \emptyset$. This is also very likely to be the case if **a** is chosen at random.

We approximate the j value at each step of KSA by a pseudorandom number and make the usual independence assumptions throughout the proofs.

4.3.2 Deriving probabilities for S table entries after KSA

The KSA changes $S[k]$ values in a structured way. This is shown by the following Lemma.

Lemma 4 *Let* $0 \leq k \leq 255$ *be an index in S. Then*

(a) $S_0[k], S_1[k], \dots S_k[k] \in W_1$

(b) $S_{k+1}[k] \in W_{n+1}$, *for each* n *such that* $S_k[j_{k+1}] \in W_n$

(c) If there exists $k+2 \leq t \leq 256$ *such that* $j_t = k$, *let* t_0 *denote the smallest such number. Then,* $S_{k+2}[k], \dots S_{t_0-1}[k] \in W_{n+1}, S_{t_0}[k], \dots S_{256}[k] \in W_1$. *If not, then* $S_{k+2}[k], \dots S_{256}[k] \in W_{n+1}$

Proof: Statement (a) is proved by induction on k. For $k = 0$, the statement takes the form $S_0[0] \in W_1$, which holds since $S_0[0] = a_0 \in W_1$. Suppose statement (a) holds for some $0 \leq k \leq 254$, we prove that (a) also holds for $k+1$. Due to the KSA procedure specification, table S_{k+1} differs from S_k only in values at indices j_{k+1} and k. As for index j_{k+1}, according to the swap step and due to induction hypothesis, we have $S_{k+1}[j_{k+1}] = S_k[k] \in W_1$. As for index k, content of $S_{k+1}[k]$ has no relevancy for the statement. Thus, (a) holds for each $k = 0 \dots 255$.

To prove (b), note that $S_{k+1}[k] = S_k[k] + S_k[j_{k+1}]$. According to (a), $S_k[k] \in W_1$. According to (b) assumption, for some n, $S_k[j_{k+1}] \in W_n$. Thus, $S_{k+1}[k] \in W_{n+1}$.

As for statement (c), suppose first that there does not exist t from the assumption of the statement. Due to (b), we have $S_{k+1}[k] \in W_{n+1}$. According to the above assumption no upcoming j_t will take value k. This value remains unchanged until the end of KSA, i.e., $S_{k+2}[k], \ldots S_{256}[k] \in W_{n+1}$. Now assume that there exists t from the assumption of (c) and let t_0 be the smallest such number. At steps $k+2, \ldots t_0 - 1$, S value on index k will not be changed and thus $S_{k+2}[k], \ldots S_{t_0-1}[k] \in W_{n+1}$ holds. However, in step t_0, value at index k is already overwritten by $S_{t_0-1}[t_0 - 1]$. According to (a), this value is a member of W_1 and thus $S_{t_0}[k], \ldots S_{256}[k] \in W_1$ also holds. $\square$

Our goal is to show that probability of $S_{256}[k] \in W_1 \cup W_2 \cup W_3 \cup W_4$ is high. First, we estimate lower bounds for the probabilities $P[S_{256}[k] \in W_n]$, $n = 1, 2, 3, 4$ separately. To do this, we note that each KSA execution in our model uniquely corresponds to a tuple $(j_0, j_1, \ldots j_{255})$ and thus can be identified with it. Lower bounds are obtained by counting tuples that, according to Lemma 4, certainly yield a value $\in W_n$ at some index k.

In the Lemmas below, we also compute $P[S_i[k] \in W_n], n = 1, 2, 3, 4$ at some of the steps $i \neq 256$. This is necessary because, as can be seen from the obtained formulas, $P[S_{256}[k] \in W_n]$, $n = 2, 3, 4$ depend on some of the $P[S_i[k] \in W_{n-1}]$.

Lemma 5 *Let i and k be the KSA step and S table index such that $i \geq k + 1$. Then,*

$$P[S_i[k] \in W_1] \geq 1 - (255/256)^{i-k-1}$$

Proof: For $i = k + 1$, the statement holds trivially since the right side of the inequality is equal to 0. Let $i \geq k + 2$. Part (c) of Lemma 4 provides a sufficient condition for event

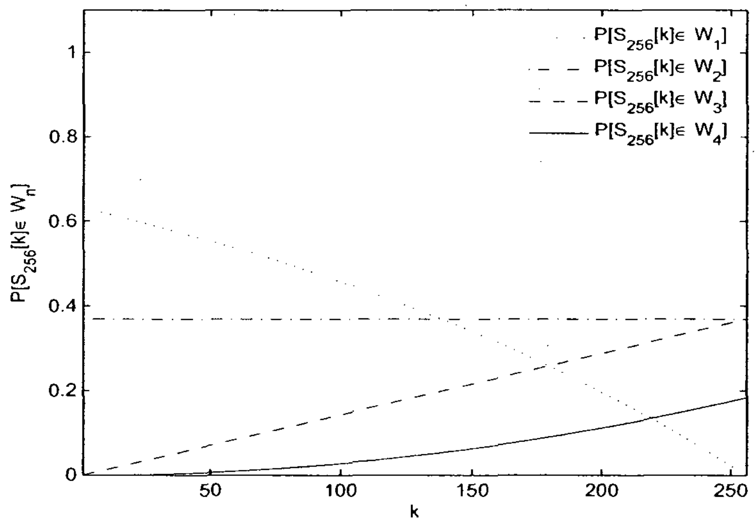

Figure 7: Probabilities that, after KSA, $S[k]$ will be representable as a sum of 1, 2, 3 or 4 values from a set

$S_i[k] \in W_1$ to hold. Namely, if for some of the $t = k+2, \ldots i$, we have $j_t = k$, $S_i[k] \in W_1$ will hold true. Thus, we can derive a lower bound for the probability of event in question as follows:

$$
\begin{aligned}
&P[S_i[k] \in W_1] \geq \\
&\quad = P[j_{k+2} = k \text{ or } j_{k+3} = k \text{ or} \ldots \text{or } j_i = k] \\
&\quad = 1 - P[j_{k+2} \neq k, j_{k+3} \neq k, \ldots j_i \neq k] \\
&\quad = 1 - P[j_{k+2} \neq k] \times P[j_{k+3} \neq k] \times \ldots \times P[j_i \neq k] \\
&\quad = 1 - (255/256)^{i-k-1}
\end{aligned}
$$

□

Lemma 6 *Let i and k be the KSA step and S table index such that $i \geq k+1$. Then,*

$$P[S_i[k] \in W_2] \geq \left(\frac{\sum_{l=0}^{k-1} P[S_k[l] \in W_1]}{256} + \frac{256-k}{256}\right) \times \left(\frac{255}{256}\right)^{i-k-1}$$

where we take that $\sum_{l=0}^{k-1} P[S_k[l] \in W_1] = 0$ *when* $k = 0$.

Proof: Similar to the proof of previous Lemma, we find a lower bound for $P[S_i[k] \in W_2]$ by noting that a combination of parts (b) and (c) of Lemma 4 provide a sufficient condition for the event in question. Namely, according to part (b) of Lemma 1, if $S_k[j_{k+1}] \in W_1$, $S_{k+1}[k] \in W_2$ will hold true. According to (c), this will remain so until step i if by then none of j indices takes value k. Thus,

$$\begin{aligned}
&P[S_i[k] \in W_2] \geq \\
&\quad = P(S_k[j_{k+1}] \in W_1, j_{k+2} \neq k, \dots, j_i \neq k) \\
&\quad = P(S_k[j_{k+1}] \in W_1) \times P(j_{k+2} \neq k), \times \dots \times P(j_i \neq k) \\
&\quad = (P(j_{k+1} < k) \times P(S_k[j_{k+1}] \in W_1 | j_{k+1} < k) + \\
&\quad\; P(j_{k+1} \geq k) \times P(S_k[j_{k+1}] \in W_1 | j_{k+1} \geq k)) \times (255/256)^{i-k-1} \\
&\quad = (\tfrac{k}{256}(\tfrac{1}{k} \times P[S_k[0] \in W_1] + \dots + \tfrac{1}{k} \times P[S_k[k-1] \in W_1]) + \\
&\quad\; \tfrac{256-k}{256}) \times (255/256)^{i-k-1} \\
&\quad = (\tfrac{\sum_{l=0}^{k-1} P[S_k[l] \in W_1]}{256} + \tfrac{256-k}{256}) \times (\tfrac{255}{256})^{i-k-1}
\end{aligned}$$

If $k = 0$, taking $\sum_{l=0}^{k-1} P[S_k[l] \in W_1] = 0$ is justified by $P[j_{k+1} < k] = 0$. In that case, we have $P[S_i[0] \in W_2] = (\frac{255}{256})^{i-1}$ for every $i \geq 1$. □

Lemma 7 *Let* i *and* k *be the KSA step and* S *table index such that* $i \geq k + 1$. *Then,*

$$P[S_i[k] \in W_3] \geq \frac{\sum_{t=0}^{k-1} P[S_k[t] \in W_2]}{256} \times (\frac{255}{256})^{i-k-1}$$

where we take that $\sum_{i=0}^{k-1} P[S_i[k] \in W_2] = 0$ *when* $k = 0$.

Proof: As in the proofs of the previous two Lemmas, the event in question can be lower bounded by events for which the probability can be easily calculated. Namely, according to parts (b) and (c) of Lemma 1, if $S_k[j_{k+1}] \in W_2$, $S_{k+1}[k] \in W_3$ will hold true and this will remain so until step i if by then none of j indices take value k.

$$P[S_i[k] \in W_3] \geq$$
$$= P(S_k[j_{k+1}] \in W_2, j_{k+2} \neq k, \ldots, j_i \neq k)$$
$$= P(S_k[j_{k+1}] \in W_2) \times P(j_{k+2} \neq k) \times \ldots \times P(j_i \neq k)$$
$$= (P(j_{k+1} < k) \times P(S_k[j_{k+1}] \in W_2 | j_{k+1} < k) +$$
$$P(j_{k+1} \geq k) \times P(S_k[j_{k+1}] \in W_2 | j_{k+1} \geq k)) \times (\tfrac{255}{256})^{i-k-1}$$
$$= \tfrac{k}{256}(\tfrac{1}{k} \times P[S_k[0] \in W_2] + \ldots + \tfrac{1}{k} \times P[S_k[k-1] \in W_2]) \times (255/256)^{i-k-1}$$
$$= \frac{\sum_{i=0}^{k-1} P[S_k[i] \in W_2]}{256} \times (\tfrac{255}{256})^{i-k-1}$$

When $k = 0$, taking $\sum_{l=0}^{k-1} P[S_k[l] \in W_2] = 0$ is justified by $P[j_{k+1} < k] = 0$ and $P[S_k[j_{k+1}] \in W_2 | j_{k+1} \geq k] = 0$. In that case, we have $P[S_i[0] \in W_3] = 0$ for every $i \geq 1$.

□

Lemma 8 *Let i and k be the KSA step and S table index such that $i \geq k + 1$. Then,*

$$P[S_i[k] \in W_4] \geq \frac{\sum_{t=0}^{k-1} P[S_k[t] \in W_3]}{256} \times (\frac{255}{256})^{i-k-1}$$

where we take that $\sum_{i=0}^{k-1} P[S_i[k] \in W_2] = 0$ when $k = 0$.

Proof: Analogous to proof of previous Lemma. □

Figure 7 illustrates the lower bounds obtained by Lemmas 2-5 for S values at indices $k = 0..255$ after the KSA has finished. Since the probability that a random number is an element of W_1, W_2, W_3 and W_4 is $\frac{|W_1|}{2^{32}} = \frac{256}{2^{32}} \approx 5.96 \times 10^{-8}$, $\frac{|W_2|}{2^{32}} = \frac{32892}{2^{32}} \approx 7.66 \times 10^{-6}$, $\frac{|W_3|}{2^{32}} = \frac{2792954}{2^{32}} \approx 0.00065$ and $\frac{|W_4|}{2^{32}} = \frac{118090021}{2^{32}} \approx 0.0275$ respectively, it is obvious that S table is biased. Sets W_n, $n \geq 5$ were not considered since the probability that a random number is contained in these sets is relatively high and hence the corresponding distinguishers would be useless.

Note that even though in Lemmas 2-5 we only estimated lower bounds, obtained values are very close to exact ones. This is because the events by which we approximated $S_i[k] \in W_n$ in the proofs of the Lemmas exhaust most of the event space. For example, in Lemma 2, event $S_i[k] \in W_1$ was lower bounded by the event that KSA operations will write one of the **a** values at place k at some point in time. The only other way $S_i[k] \in W_1$ event can hold is that KSA leaves an addition of **a** values at place k and this addition turns out to be an element of W_1 too, which is possible since $W_i, i = 1, 2 \ldots$ are not generally disjoint. However, the probability of this is very small and thus our estimated lower bound is actually a good estimate for exact $P[S_i[k] \in W_1]$ value. Similar reasoning applies for Lemmas 3, 4 and 5. Using $10,000$ randomly generated 128-bit keys, the experimentally calculated values of $P[S_i[k] \in W_n], n = 1, 2, 3$ confirmed the above claim.

4.4 Distinguishing NGG from a random stream

As noted in previous section, the probability that $S_{256}[k] \in W_n$, $n = 1, 2, 3, 4$ is significantly greater than the probability that some random number will be an element of these

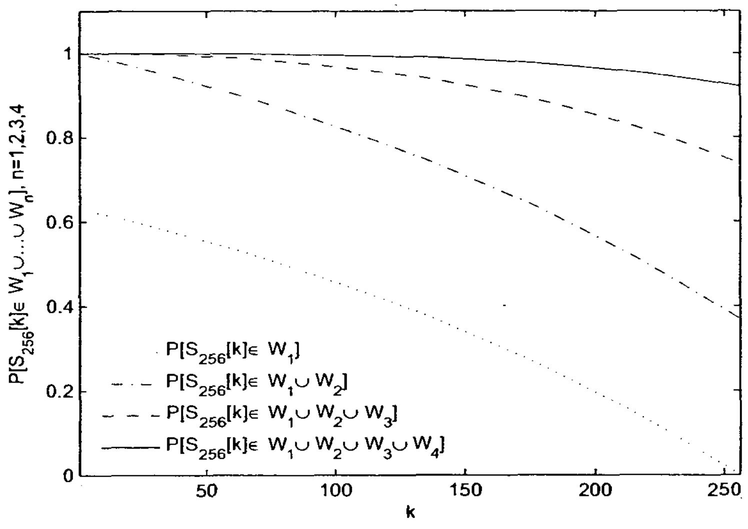

Figure 8: Probability that after KSA S[k] will be an element of different unions of W sets

sets. However, to maximize this bias, we consider the union of these sets. Figure 8 illustrates the probability that after the KSA, the S values will be contained in different unions of W sets. Depending of which sets are included in the union, different distinguishing criteria can be formulated. As will be shown, the probability that a random number will be in any of these unions is still small.

In the following theorem, we determine a lower bound on the success probabilities of the distinguishers using criterions based on different unions of the $W_i, i = 1, 2, 3. 4$ sets.

Theorem 3 *Using the first NGG PRGA keystream output word, k_1, lower bound of success probability of distinguishing k_1 from a random stream are given in Table 1 below.*

Proof: First, we prove the statement for the distinguisher 3, in which the distinguisher is based on $k_1 \in W_1 \cup W_2 \cup W_3$ criterion. Proofs for distinguisher 1 and 2 are similiar. Suppose that the given word, o, is the first NGG PRGA output word, i.e., $o = k_1 = S_{256}[k]$

#	*Distinguishing criterion*	*P[Correct decision]*
1	$k_1 \in W_1$	*0.6836*
2	$k_1 \in W_1 \cup W_2$	*0.8679*
3	$k_1 \in W_1 \cup W_2 \cup W_3$	*0.9597*
4	$k_1 \in W_1 \cup W_2 \cup W_3 \cup W_4$	*0.9769*

Table 1: Lower bounds on the success rate of different distinguishers

for some k. Then, lower bound of the probability that it will be an element of $W_1 \cup W_2 \cup W_3$ can be calculated by applying the Bayes' formula over all possible values k, using the fact that sets W_1, W_2 and W_3 are mutually disjoint and substituting according values to lower bounds from Lemmas 2-4.

$$P[S_{256}[k] \in W_1 \cup W_2 \cup W_3] = \tag{18}$$

$$\sum_{i=0}^{255} P[k = i] P[S_{256}[k] \in W_1 \cup W_2 \cup W_3 | k = i] = \tag{19}$$

$$\frac{1}{256} \sum_{i=0}^{255} (P[S_{256}[i] \in W_1] + P[S_{256}[i] \in W_2] + P[S_{256}[i] \in W_3]) \tag{20}$$

$$\geq 0.92$$

Thus, in the case that the word o is in fact an NGG first keystream word, probability that the distinguisher will make a right decision is greater than 0.92. In the case of a random word, the probability that it will not be representable as a sum of one, two or three elements of $\mathbf{a}$ is given by

$$1 - \frac{|W_1 \cup W_2 \cup W_3|}{2^{32}} \approx 0.9993.$$

Thus, the success probability of distinguisher based on $k_1 \in W_1 \cup W_2 \cup W_3$ criterion will be greater than $0.5 \times 0.92 + 0.5 \times 0.9993 = 0.9597$.

As for distinguisher 4, event $S_{256}[k] \in W_1 \cup W_2 \cup W_3 \cup W_4$ is not a disjoint union of events $S_{256}[k] \in W_1$, $n = 1, 2, 3, 4$ and thus going from (19) to (20) in the corresponding proof would not be justified. However, the lower bound events established in the proofs of Lemmas 2-5 are mutually disjoint. Let L_1 be the lower bound event used in Lemma 2 for $i = 256$ $L_1 = \{j_{k+1} = k \text{ or } j_{k+2} = k \text{ or } \ldots \text{ or } j_{255} = k\}$ and let events L_2, L_3 and L_4 be the corresponding events from Lemmas 3-5. Then, we have $P[S_{256}[k] \in W_1 \cup W_2 \cup W_3 \cup W_4] \geq P[L_1 \cup L_2 \cup L_3 \cup L_4] = P[L_1] + P[L_2] + P[L_3] + P[L_4]$. The proofs of Lemmas 2-5 provide exact values for $P[L_1]$, $P[L_2]$, $P[L_3]$, $P[L_4]$. The rest of the proof is analogous to previous ones. $\square$

4.4.1 Distinguishers based on the m-th keystream word

In our reasoning above, we only considered the first keystream word. Calculated probabilities apply only to this word since the S table is updated at each step. However, it is not hard to see that the same distinguisher is applicable to the m-th keystream word k_m too, provided that m is small, but with decreasing success rate as m increases.

To assess success rate of the distinguisher $k_m \in W_1 \cup W_2 \cup W_3$, we conducted the following experiment. For 10000 times, the NGG cipher was initialized by KSA for randomly generated key. The percentage of k_m values which were not representable as a sum of one, two or three a numbers was taken to be the probability of false negatives for distinguisher based on k-th element. Table 2 presents results for some m values. It can be seen that distinguisher guess still differs from random guess at $m = 1024$. This implies that the traditional solution to resist distinguishing attacks against RC4-like cipher by discarding some of the output stream words may not practicably work in this case. Naturally, the success of the distinguisher can be arbitrarily increased by looking at several keystream words at once.

m	16	32	64	128	256	512	1024
P[Error]	0.050	0.066	0.077	0.150	0.288	0.407	0.484

Table 2: Probability of error for distinguishers based on the m-th keystream word

4.5 Key information recovery

In this section, we show how to use the observations above to reduce the entropy of the the secret key based on the first few kilobytes of the keystream.

From the algorithm specifications, it is clear that knowing $j_0, j_1, \dots j_{16}$, i.e., the values assumed by the pseudorandom counter j in the first 16 rounds of the scrambling step of KSA, is equivalent to knowing all key bytes $K_0, \dots K_{15}$. Thus, it suffices to focus on reducing the entropy of the vector $(j_0, j_1, \dots j_{16})$.

4.5.1 Recovering the first secret key byte

Among the first 1024 keystream words, all entries of table S_{256} will be present with high probability. The reason for this is that the NGG PRGA sends the S table entries to the output unmasked and changes them only once they are outputted.

As mentioned above, to determine K_0, it suffices to find j_1 since $K_0 = j_1 - 0 - \mathbf{a}_0$. At the first step of the KSA scrambling step, $S[0] + S[j_1] = \mathbf{a}_0 + \mathbf{a}_{j_1}$ is written to $S[0]$. The only way $S[0]$ can be changed again is that that for some $i \geq 2, j_i = 0$. The probability that this will not happen, i.e., $S[0]$ will remain $\mathbf{a}_0 + \mathbf{a}_{j_1}$ is $(255/256)^{255} = 0.3686$. By trying to decompose each of the first 1024 keystream words in the form of $\mathbf{a}_0 + \mathbf{a}_x$, with probability 0.3686, one of the obtained x values will be equal to j_1. However, there might be more than one $\mathbf{a}_0 + \mathbf{a}_x$ element in S table. Namely, at the first KSA scrambling step, the value $S[0] = \mathbf{a}_0$ is also written to $S[j_1]$. At the j_1-th scrambling step, $S[j_1]$ will be added another element from the table. If $S[j_1]$ is not altered again by any of remaining steps, there will be two $\mathbf{a}_0 + \mathbf{a}_x$ elements in the table and there would be two candidates for j_1.

Similar reasoning applies for the case when, after first scrambling step of KSA, the value $S[0]$ is altered again by some j_i. Then, a value of the form $\mathbf{a}_0 + \mathbf{a}_{x_1} + \mathbf{a}_{x_2}$ will be present in the table after the KSA finishes, provided that it is not altered again. The algorithm can search for values of the form $\mathbf{a}_0 + \mathbf{a}_{x_1}$, $\mathbf{a}_0 + \mathbf{a}_{x_1} + \mathbf{a}_{x_2}$, $\mathbf{a}_0 + \mathbf{a}_{x_1} + \mathbf{a}_{x_2} + \mathbf{a}_{x_3}$ and $\mathbf{a}_0 + \mathbf{a}_{x_1} + \mathbf{a}_{x_2} + \mathbf{a}_{x_3} + \mathbf{a}_{x_4}$ among the first 1024 keystream words. There is a tradeoff between number of candidates and the corresponding success probability. In the following section, we experimentally evaluate a full key recovery algorithm based on first two forms.

4.5.2 Full key recovery algorithm

To derive a candidate for j_t, $t = 2, \ldots 16$, we use the same idea, only now we have smaller probability of success since $S_{t-1}[t-1]$ and $S_{t-1}[j_t]$ might not be equal to $\mathbf{a}_{t-1}$ and $\mathbf{a}_{j_t}$, respectively. The probability that this will hold decreases as t increases.

Let $cand(J_i)$ denote the set of candidates for value j_i. Let $cand(J) = cand(J_0) \times \ldots \times$

```
INPUT: First 1024 NGG keystream words
OUTPUT: A set of secret key candidates (cand(K))
 1: Let $cand(J_0) = \{0\}$ and $cand(J_i) = \emptyset$ for $1 \le i \le 16$
 2: For each word $w$ of first 1024 keystream words do
 3:     For $i = 1, \ldots 16$, do
        - Check if $w$ can be written as $\mathbf{a}_{i-1} + \mathbf{a}_{x_1}$, where $x_1 \in \{0, 1, \ldots 255\}$.
          If yes, add $x_1$ to $cand(J_i)$
        - Check whether $w$ can be written as $\mathbf{a}_{i-1} + \mathbf{a}_{x_1} + \mathbf{a}_{x_2}$, where
          $x_1, x_2 \in \{0, 1, \ldots 255\}$. If yes, add $x_1$ and $x_2$ to $cand(J_i)$
 4: $cand(J) = cand(J_0) \times cand(J_1) \times \ldots \times cand(J_{16})$
 5: Return $cand(K) = f(cand(J))$
```

Figure 9: NGG key recovery algorithm

$cand(J_{16})$ denote the set of candidates for the vector $j = (j_0, \ldots j_{16})$. Let $f((j_0, \ldots j_{16})) = (k_0, \ldots k_{15})$ where $k_i = (j_{i+1} - j_i - \mathbf{a}[i]) \bmod 256$. The set of candidates for key K can be obtained as $cand(K) = f(cand(J))$.

Figure 9 shows the algorithm which recovers $cand(J)$, and consequently the set of secret key candidates, based on the first 1024 keystream words.

It should be noted that computing $\mathbf{a}_{i-1} + \mathbf{a}_{x_1}$, $\mathbf{a}_{i-1} + \mathbf{a}_{x_1} + \mathbf{a}_{x_2}$ for each i, x_1 and x_2 is repeated in each loop pass. Thus, the algorithm can be efficiently executed in negligible time if these values are precomputed offline and sorted. However, as will be seen, the number of candidates for secret key K that the algorithm yields is close to 2^{32} and to discard wrong candidates, a computation of around 2^{32} operations cannot be avoided.

Since the function f is 1-1, only the correct j vector candidate will be mapped to the correct key. Thus, the effectiveness of proposed algorithm can be measured as follows:

- $P[(k_0, \ldots k_{15}) \in cand(K)] = P[(j_0, \ldots j_{16}) \in cand(J)]$, i.e., the probability that correct key, or equivalently, j values, will be found. For some particular $i \in \{1, \ldots 16\}$,

$j_i \in cand(J_i)$ will hold if at most 2 **a** values are added to $\mathbf{a}_i$ which is placed in $S_0[i]$ at the beginning of KSA scrambling.

- $|cand(K)| = |cand(J)|$, i.e., the number of key candidates. For some particular $i \in \{1, \dots 16\}$, $|cand(J_i)|$ depends on the number of interactions between $S_0[i] = \mathbf{a}_i$ and other **a** values throughout the KSA. Since the additions continue during the PRGA, wrong candidates not present in table at moment $i = 256$ might emerge during 1024 PRGA steps performed during the algorithm.

Deriving an analytical expression for $P[j_i \in cand(J_i)]$ and $|cand(J_i)|$ seems to be a relatively hard combinatorial problem. Thus, in order to test the success rate of the algorithm, the following experiment was conducted. For 1000 randomly generated keys, sets $cand(J_i)$, $i = 1, \dots, 16$ were calculated according to the algorithm above. Percentage of times $j_i \in J_i$ was true and average $|J_i|$ values are shown in Table 3. Value j_0 is omitted because it is always equal to 0. For example, according to the experiment above, the correct

i	1	2	3	4	5	6	7	8
$P[j_i \in cand(J_i)]$	0.7223	0.7373	0.6895	0.7242	0.6848	0.7221	0.6885	0.6913
$\|cand(J_i)\|$	3.9362	4.0994	4.0225	4.0629	3.9268	3.9740	3.9510	3.8712
i	9	10	11	12	13	14	15	16
$P[j_i \in cand(J_i)]$	0.7000	0.6577	0.6558	0.6538	0.6702	0.6462	0.6375	0.6578
$\|cand(J_i)\|$	4.0837	3.7904	3.9808	3.8288	3.8221	3.8365	3.8144	3.9250

Table 3: Success rate and average number of candidates for j_i, $1 \leq i \leq 16$

j_1 value will be among on average 3.9362 candidates proposed by the algorithm with probability 0.7223. Since $j_0 = 0$ by algorithm specification, we then have K_0 will be among 3.9362 candidates with probability 0.7223. The probability that both $j_1 \in J_1$ and $j_2 \in J_2$

is 0.7223×0.7373 and in that case, both K_0 and K_1 will be among proposed candidates. To generalize, $K \in cand(K)$ with probability $\approx 2^{-8.8}$ (obtained by multiplying all the probabilities in Table 3) and $|cand(K)| \approx 2^{31.6}$ (obtained by multiplying the number of candidates in Table 3). In other words, the above result can be restated as follows: for approximately $2^{119.2}$ of the 128-bit keys, the secret key can be recovered with around 2^{32}, instead of 2^{128}, steps of computation.

Chapter 5

On the Weak State of GGHN-like Ciphers

In this chapter we consider another RC4 generalization, the GGHN stream cipher. As mentioned in the previous chapter, RC4 is a cipher that makes use of an internal state table, S, which represents a permutation over Z_{2^8}. Like NGG, GGHN is a relatively more efficient stream cipher whose design is inspired from RC4 but whose S table, however, does not represent a permutation over Z_{2^m}. In this chapter, we point out one challenging aspect of the latter design principle. In particular, we assess different variants of GGHN-like algorithms with respect to weak states, in which all internal state words and output elements are even. Once GGHN is absorbed in a weak state, the least significant bit of the plaintext words will be revealed only by looking at the ciphertext. By modelling the algorithm by a Markov chain and calculating chain's absorption time, we show that the average number of steps required by these algorithms to enter this weak state can be lower than expected at first glance and hence caution should be exercised when estimating this number.

5.1 Introduction

As mentioned in the previous chapter, RC4 operates with 8-bit keystream words, which is inefficient from the perspective of 32 and 64 bit processors. RC4-like cipher that would produce 32-bit or 64-bit keystream words would be around 4-8 times faster, since an 8 bit operation on these processors takes equal time as a 32, or 64 bit operation. To address this problem, Gong *et al.* [50] first proposed the cipher analyzed in the previous chapter of this thesis, a generalized version of RC4, namely RC4(n,m), where m denotes the bit-length of the keystream output word. This cipher was later named NGG. Later, GGHN cipher [23] was introduced.

The internal state of RC4 includes an S table of 256 8-bit values. The important property of this table is that, at each step, it represents a permutation, i.e., all the possible combinations of the m bit values appear in the table. In NGG(n,m) and GGHN(n,m), the size of the table that would be needed to store the whole permutation is 2^m where m is the keystream word size, which is impractical for $m > 8$. Accordingly, smaller table of size N had to be used in both versions of the ciphers where $N = 2^n << 2^m$ represents the table size and m is the keystream word size in bits. Since a table can now hold only a subset of the possible m bit values, to avoid distinguishing attack which would test whether keystream words belong to only a subset of all possible m-bit values, internal state update function had to not only perform swap operation as in the original RC4 but to introduce new values to the S table. In NGG, this was done by replacing the S table element chosen as keystream word by a sum of two other elements from the table at each step. However, this was proven to be susceptible to an attack [66] and GGHN(n,m) was proposed. In GGHN(n,m) another counter, k, was introduced and attacks similar to [66] are supposedly eliminated.

5.1.1 Our contribution

By weak state of GGHN, we denote a state in which k and all the S table entries are even. Since the update and output functions are defined as additions of these elements, once the cipher reaches this weak state, it will remain in it forever and the output keystream words will always be even. Therefore, once GGHN is in a weak state, the least significant bit of the plaintext words will be revealed only by looking at the ciphertext.

In [23], it was noted that the probability that all state entries as well as k become even is very low, $2^{-(N+1)}$, which implies that the average number of steps before GGHN(n,m) enters weak state is $O(2^N)$. In this chapter we revisit this claim and provide evidence that caution should be taken when estimating this number for GGHN-like ciphers. We also show that, if the designers of GGHN kept the swap operation (originally present in both RC4 and NGG ciphers) for the S table scrambling during the pseudorandom generation algorithm, then the cipher would enter the weak state in $O(2^{0.5N})$ steps as opposed to the $O(2^N)$ steps that might be expected when treating the S table as a purely random table.

Since the exact behavior of GGHN cipher with respect to the number of even numbers in the table is hard to model due to non Markovian nature of the induced chains, we decided to use a simplified version of GGHN and show that it enters weak state faster than expected at first glance. We model this simplified version of GGHN by a Markov chain and estimate the average number of steps before reaching the weak state using results from Markov chain theory.

Based on our analysis, it follows that with GGHN-like ciphers that are based on non bijective S tables, caution should be made when estimating the time needed for approaching weak state.

The rest of the chapter is organized as follows. In section 2, we review the specifications of RC4 and GGHN ciphers. Section 3 gives an overview of some attacks on these ciphers. In section 4, we provide a brief theoretical background for Markov chains. In section 5, an idealized model of GGHN-like ciphers is presented and then modelled by a Markov chain. Section 6 presents a way to calculate the absorption times for the defined chain. In section 7 we provide experimental results that indicate that original GGHN(n,m) may enter weak state faster than previously assumed. Finally, the conclusion is given in section 8.

5.2 RC4 and GGHN specification

In both ciphers, the size of the key is l bytes. Usually, $l = 8$ or $l = 16$. The RC4 KSA procedure initializes the S array to an identity permutation and then swaps each byte with a pseudorandom element of the table. In RC4 PRGA, two counters i and j are used. Counter i is publicly known and j is incremented in a pseudorandom way. At each step, element

KSA	PRGA
Initialization:	*Initialization:*
For $i = 0, ..255$	$i = j = 0$
$S[i] = i$	*Loop:*
$j = 0$	$i = i + 1$
Scrambling:	$j = j + S[i]$
For $i = 0, ..255$	Swap$(S[i], S[j])$
$j = j + S[i] + K[i \bmod l]$	$t = S[i] + S[j]$
Swap$(S[i], S[j])$	Output $z = S[t]$

Table 4: RC4 specification

$S[i]$ and $S[j]$ are swapped. Element $S[S[i] + S[j]]$ is chosen as the output keystream byte. A pseudo code for RC4 is shown in Fig. 4.

In GGHN(n,m), another pseudorandom counter, k, is added to the algorithm. Again, the S table size is $N = 2^n$ and the internal and keystream word size is m bits. The GGHN KSA is similar to the NGG KSA with the only difference is that it is repeated r times (it is recommended to set $r = 20$ for $n = 8$). As for k, after it is initialized to 0 at the beginning of KSA, the S table entries are added to it iteratively. In comparison to the NGG PRGA, the GGHN PRGA makes use of a pseudorandom counter, k, both in the update and the output steps. Instead of outputting plain pseudorandom entry of the S table as in NGG, value masked by k will constitute the keystream word. Similarly, new value that will be written to the array depends on k. The pseudo code of the algorithm is shown in Fig. 5.

KSA

```
Initialization:
For $i = 0,..N-1$
  $S[i] = a_i$
$j = k = 0$
Scrambling:
 Repeat $r$ times
  For $i = 0,..N-1$
    $j = (j + S[i] + K[i \bmod l]) \bmod N$
    Swap($S[i], S[j]$)
    $S[i] = (S[i] + S[j]) \bmod M$
    $k = (k + S[i]) \bmod M$
```

PRGA

```
Initialization:
$i = 0$
$j = 0$
Loop:
  $i = (i + 1) \bmod N$
  $j = (j + S[i]) \bmod N$
  $k = (k + S[j]) \bmod M$
  $t = (S[i] + S[j]) \bmod N$
  out=$(S[t] + k) \bmod M$
  $S[t] = (k + S[i]) \bmod M$
```

Table 5: GGHN(n,m) specification

5.3 Existing attacks

Security of RC4 has been a subject of extensive research. In this section, we provide a brief summary of some of these results as well as existing attacks on GGHN.

5.3.1 Attacks on RC4

Knudsen *et al.* [33] proposed a backtracking algorithm for RC4 internal state recovery. The attack guesses some values from the internal state, simulates the update process and searches for contradiction between the keystream output and known values. If a contradiction is met, the algorithm backtracks to the last guessed values and chooses another one. It requires a small portion of starting keystream word, but has unrealistic time complexity which amounts to more than 2^{700} steps. In [42], an attack on RC4 that requires 2^{241} operations and similar amount of data is presented. The given algorithm is based on searching for keystream patterns that correspond to certain internal states.

As for distinguishing attacks, Golić [22] shows that the sum of least significant bits of two keystream words at times t and $t+2$ is biased towards 1. To distinguish RC4 from random variable, around 2^{40} keystream words are needed.

A more efficient distinguisher was found in 2000, when Fluhrer *et al.* [18] showed that there exists strong correlation between digraphs of RC4 (i.e., consecutive output bytes). This correlation permits an attacker to distinguish RC4 keystream from a random output in around $2^{30.6}$ steps.

In 2001, an unexpected discovery regarding bias of first two output words was made by Mantin and Shamir [36]. Instead of looking at the keystream bytes of one key, this distinguisher makes a decision by looking at first two keystream bytes under a few hundred of different, unrelated and unknown keys. The authors used this idea to mount an attack against the broadcast version of RC4, in which they recover second plaintext byte based only on ciphertext.

As shown by Fluhrer *et al.*, RC4 is insecure when known IV value is concatenated to a key [19]. This mode of operation is used in Wired Equivalent Privacy Protocol (WEP). Mantin [38] showed that WEP can be attacked this way even if the first 256 bytes of the keystream are discarded. In 2006, Klein [29] presented an improved known plaintext attack which does not need weak IVs to recover the secret key. Vaudenay and Vuagnoux improved on previous attacks using the weakness they observed in KSA [62]. Tews *et al.* found an attack on a 104 bit WEP [59].

Mantin [37] found a new distinguishing attack based on biases by which digraphs tend to repeat with short gaps between them. The attack needs around 2^{26} keystream words and has success rate around $2/3$. In the same chapter, it is shown that RC4 keystream output has a family of patterns that repeat themselves with probability several times higher than in truly random sequences. These patterns can be used to predict bits and words of RC4, using 2^{45} and 2^{50} keystream words, respectively.

Applying theory of random shuffles, Mironov [47] has modeled RC4 by a Markov chain and recommended that at least 512 beginning keystream bytes should discarded in order for S table to be uniformly distributed. This way, any attacks based on non-randomness of the beginning S table should be circumvented.

5.3.2 Attacks on GGHN

Just as with NGG, the least significant bit of the keystream output is biased, due to a bias inducing state which occurs with probability 2^{-16} and induces that the LSB of keystream words is 0 with certainty [51]. The distinguisher can be built upon $2^{32.89}$ keystream words.

Tsunoo *et al.* [61] presented a distinguisher based on the first two words of keystream associated with approximately 2^{30} keys.

5.4 Markov Chains background

In this section we briefly review the Markov chains, absorption states and a way to calculate the average absorption times from the chain transition matrix [24].

A Markov chain can be described as follows. Let $S = \{s_1, s_2, \ldots s_n\}$ denote the set of *states*. The chain starts in some state, and in each step, moves from one state to another.

The probability that the chain will move to certain state s_j depends only on its previous state s_i, and not on any of the states before that. Therefore, we can write p_{ij} for transition probabilities, denoting the probability that a process will go from s_i to s_j. The matrix $P = (p_{ij})_{r \times r}$ is called the transition matrix. The beginning state of the chain is usually described by a probability distribution over the set S. The following lemma is a rudimentary result on Markov chains.

Lemma 9 *Let P be a transition matrix of a Markov chain with set of states $S = \{s_1, s_2, \ldots s_r\}$. The p_{ij} entry of matrix P^n then represents the probability that the chain, starting in state s_i, will be in s_j after n steps.*

When a pseudorandom number generator that we are modelling by a Markov chain enters the weak state, it cannot go out of it anymore. This behavior corresponds to absorbing state notion, known in Markov chain theory.

Definition 2 *A state s_i of a Markov chain is called absorbing state if $p_{ii} = 1$, i.e., if once entered, it is impossible for the chain to leave it. A Markov chain is called absorbing if it has at least one absorbing state and if, from every state, it is possible to go to absorbing state (not necessarily in one step). Non-absorbing states of an absorbing chain are called transient states. When a chain enters absorption state, we say that it got absorbed.*

Lemma 10 *In an absorbing Markov chain, the probability that the process will be absorbed is 1.*

To measure the number of steps needed for absorption, it is convenient to order states from set S so that absorbing states come at the end. Suppose that there are t transient states and r absorbing states. Then, the transition matrix will be of the form:

$$\left[\begin{array}{c:c} \mathbf{Q} & \mathbf{R} \\ \hdashline \mathbf{0} & \mathbf{Id} \end{array}\right]$$

where **Id** is the identity matrix and **0** is zero matrix. The dimensions of **Q**, **R**, **0** and **Id** are $t \times t$, $t \times r$, $r \times t$ and $r \times r$, respectively. The matrix **I-Q** is called the *fundamental matrix* and its inverse reveals information with respect to absorption time. In particular, we use the following result.

Lemma 11 *For an absorbing Markov chain, the matrix **I-Q** is invertible. The ij-entry of $(\mathbf{I}\text{-}\mathbf{Q})^{-1}$ is the expected number of times the chain is in state s_j, given that it started from s_i. The initial state is counted if $i = j$.*

That way, the inverse of fundamental matrix gives information on how many times will a process pass through each state before absorption. We can now calculate the expected number of steps before absorption. Namely, if we sum all the values of i-th row of matrix $(\mathbf{I}\text{-}\mathbf{Q})^{-1}$, we get the expected number of steps before absorption, given that a process starts from state s_i.

5.5 Modeling idealized GGHN(n,m) by a chain

As specified in GGHN PRGA procedure, counter i iterates from 0 to $N - 1$ and j and k are incremented in a pseudorandom way. At each step, k is updated as $k = (k + S[j]) \bmod M$, $(S[(S[i] + S[j]) \bmod N] + k) \bmod M$ element is sent to the output and the S table is updated by $S[(S[i] + S[j]) \bmod N] = (k + S[i]) \bmod M$.

To model GGHN-like ciphers, we idealize it as follows. For the update step of k, we take $k = (k + S[X]) \bmod M$ where X is random variable taking values from 0 to $N - 1$, thus approximating pseudorandom number j by a uniformly distributed random value X. As for the output, we model it as $output = (S[Y] + k) \bmod M$, where Y is a uniform random variable independent from X and takes values from 0 to $N - 1$ and substituting pseudorandom value $(S[i] + S[j]) \bmod N$. Finally, for the update step, we take $S[Y] = (k + S[Z]) \bmod M$. Here we use already defined Y, and introduce Z, uniform random value independent of X and Y, as substitution for counter i. The PRGA algorithm of GGHN*(n,m) is then modelled as follows:

$$\begin{aligned} k' &= k + S[X] \bmod M \\ output &= k + S[Y] \bmod M \\ S[Y] &= k + S[Z] \bmod M \end{aligned}$$

Obviously this algorithm can only be used as an idealized model for the GGHN cipher. However, we use it for the purposes of illustrating how different variants of GGHN(n,m)-like ciphers may enter the weak state in a number of steps which is significantly smaller than 2^N. Note that we do not initialize S by any KSA, but randomly. An eventual bias towards even numbers in beginning table may amplify the results given in the chapter.

We say GGHN*(n,m) is in weak state if all elements of S as well as value k are even.

5.5.1 Defining the chain and calculating state transition probabilities

We map each possible internal state of the cipher to a state from the chain's state set. Let $Z(S)$ denote the number of odd numbers in the S table. We say GGHN*(n,m) is in state (z, p) if $Z(S) = z$ and $p = k$ mod 2 for current S and k. For example, if GGHN*(8,32) is in state (130,0), that means that exactly 130 values in S are odd, and k is even. Since $Z(S) \in \{0..N\}, p \in \{0, 1\}$, we have that for GGHN*(n,m) the number of states is equal to $2 \times (N + 1)$. It follows that GGHN* is in weak state if and only if it is in $(0, 0)$ state.

Due to the independence between X, Y and Z and other parts of internal state, this chain satisfies the Markovian property. In other words, if current state is (z_1, p_1), probability distribution of next state (z_2, p_2) does not depend on previous state (z_0, p_0) or states before that. In the following theorem, we calculate state transition probabilities.

Theorem 4 *Let state transition probabilities not mentioned in the table amount to zero. In rows 1-4 of the table, let $z = 1 \ldots N$. In rows 5-8 and 9-12 let $z = 0 \ldots N$ and $z = 0 \ldots N - 1$, respectively. Then, the table specifies state transition probabilities of the defined chain.*

State transition	*Probability*
$P[(z,0) \rightarrow (z-1,0)]$	$(\frac{N-z}{N})^2 \times \frac{z}{N}$
$P[(z,0) \rightarrow (z-1,1)]$	$(\frac{z}{N})^3$
$P[(z,1) \rightarrow (z-1,0)]$	$(\frac{z}{N})^2 \times \frac{N-z}{N}$
$P[(z,1) \rightarrow (z-1,1)]$	$\frac{N-z}{N} \times (\frac{z}{N})^2$
$P[(z,0) \rightarrow (z,0)]$	$\frac{N-z}{N} \times ((\frac{N-z}{N})^2 + (\frac{z}{N})^2)$
$P[(z,0) \rightarrow (z,1)]$	$2 \times (\frac{z}{N})^2 \times \frac{N-z}{N}$
$P[(z,1) \rightarrow (z,0)]$	$\frac{z}{N} \times ((\frac{N-z}{N})^2 + (\frac{z}{N})^2)$
$P[(z,1) \rightarrow (z,1)]$	$2 \times (\frac{N-z}{N})^2 \times \frac{z}{N}$
$P[(z,0) \rightarrow (z+1,0)]$	$(\frac{N-z}{N})^2 \times \frac{z}{N}$
$P[(z,0) \rightarrow (z+1,1)]$	$\frac{z}{N} \times (\frac{N-z}{N})^2$
$P[(z,1) \rightarrow (z+1,0)]$	$\frac{N-z}{N} \times (\frac{z}{N})^2$
$P[(z,1) \rightarrow (z+1,1)]$	$(\frac{N-z}{N})^3$

Proof: Due to the fact that only one S table element is updated at each PRGA iteration, the number of even values in the table can either decrease by one, stay the same or increase by one. Thus, all probabilities not mentioned in the table are equal to 0.

Let $P[even(\cdot)]$ and $P[odd(\cdot)]$ denote the probability that the enclosed argument is even and odd, respectively. We prove rows 1-4 of the table. Let $z = 1..N$. Then,

$$P[(z,0) \rightarrow (z-1,0)] =$$

$$P[even(S[X])] \times P[odd(S[Y])] \times P[even(S[Z])] =$$

$$\frac{N-z}{N} \times \frac{z}{N} \times \frac{N-z}{N}.$$

$$P[(z,0) \rightarrow (z-1,1)] =$$

$$P[odd(S[X])] \times P[odd(S[Y])] \times P[odd(S[Z])] =$$

$$\frac{z}{N} \times \frac{z}{N} \times \frac{z}{N}.$$

$$P[(z,1) \rightarrow (z-1,0)] =$$

$$P[odd(S[X])] \times P[odd(S[Y])] \times P[even(S[Z])] =$$

$$\frac{z}{N} \times \frac{z}{N} \times \frac{N-z}{N}.$$

$$P[(z,1) \rightarrow (z-1,1)] =$$

$$P[even(S[X])] \times P[odd(S[Y])] \times P[odd(S[Z])] =$$

$$\frac{N-z}{N} \times \frac{z}{N} \times \frac{z}{N}.$$

Now let $z = 0 \ldots N$. Then, rows 5-8 can be proven as follows.

$$P[(z,0) \rightarrow (z,0)] =$$

$$P[even(S[X])] \times (P[even(S[Y])] \times P[even(S[Z]] +$$

$$P[odd(S[Y])] \times P[odd(S[Z])]) =$$

$$\frac{N-z}{N} \times (\frac{N-z}{N} \times \frac{N-z}{N} + \frac{z}{N} \times \frac{z}{N}).$$

$$P[(z,0) \rightarrow (z,1)] =$$

$$P[odd(S[X])] \times (P[even(S[Y])] \times P[odd(S[Z])] +$$

$$P[odd(S[Y])] \times P[even(S[Z])]) =$$

$$\frac{z}{N} \times (\frac{N-z}{N} \times \frac{z}{N} + \frac{z}{N} \times \frac{N-z}{N}).$$

$$P[(z,1) \rightarrow (z,0)] =$$

$$P[odd(S[X])] \times (P[even(S[Y])] \times P[even(S[Z])] +$$

$$P[odd(S[Y])] \times P[odd(S[Z])]) =$$

$$\frac{z}{N} \times (\frac{N-z}{N} \times \frac{N-z}{N} + \frac{z}{N} \times \frac{z}{N}).$$

$$P[(z,1) \to (z,1)] =$$

$$P[even(S[X])] \times (P[even(S[Y])] \times P[odd(S[Z])] +$$

$$P[odd(S[Y])] \times P[even(S[Z])]) =$$

$$\frac{N-z}{N} \times (\frac{N-z}{N} \times \frac{z}{N} + \frac{z}{N} \times \frac{N-z}{N}).$$

Proof of rows 9-12 is analogous to the proof of rows 1-4. □

5.6 Absorption times

To calculate the absorption time for the above defined chain, we follow guidelines from section 5.4. First, we encode states (z,p) $(z = 0 \dots N, p = 0,1)$ by numbers from 0 to $2N+1$ using one-to-one function $f(z,p) = 2N - 2z + 1 - p$. Note that $f((0,0)) = 2N+1$, so in this ordering, weak state of GGHN*, in which there is 0 odd numbers in the S table and k is even, comes last, as suggested in section 5.4. The transition matrix **P** indexed from 0 to $2N+1$ in both dimensions will contain value $P[(z_0,p_0) \to (z_1,p_1)]$ (given by Theorem 4) at index (i,j), where $i = f(z_0,p_0)$ and $j = f(z_1,p_1)$. Then, discarding last row and last column gives matrix **Q** and **B=I-Q** represents the fundamental matrix. Finally we calculate $\mathbf{B}^{-1}$ and then, the sum of i-th row of this matrix represents average number of steps GGHN* will take before absorption if the algorithm started from $f^{-1}(i)$ state. We approximate the beginning state distribution by assuming that half of the numbers in S table are odd and value k is even and thus we take the sum of $f((N/2,0))$ row as an estimate for the expected absorption time of the defined chain.

Table 6 shows the match between the analytical and experimental estimate for the average number of steps before GGHN* enters the weak state. The experiment was conducted by initializing S table and k to random and running the GGHN*(4,32) ($N = 16$) and GGHN*(5,32) ($N = 32$) until reaching weak state, for 1000 times.

	Avg. number of steps	
N	Markov model	Experimental
16	$2^{12.763}$	$2^{12.813}$
32	$2^{21.691}$	$2^{21.606}$
64	$2^{39.1}$	-

Table 6: Average number of steps before GGHN*(n,m) enters weak state

One of the discrepancies between GGHN and GGHN* is the difference in distribution of $(S[i] + S[j]) \bmod N$ and its idealized substituted uniform random variable, Y. Unlike Y, $(S[i] + S[j]) \bmod N$ tends to be even more often whenever the number of even entries exceeds the number of odd entries in S. In other words, this bias results in the cipher more often outputting, and also updating, even-indexed S table entries. In fact, this observation is interesting in itself because, typically, one would expect that any bias in the S table update process would weaken the cipher. However, as will be shown in the next section, this bias results in a longer absorbtion time for GGHN.

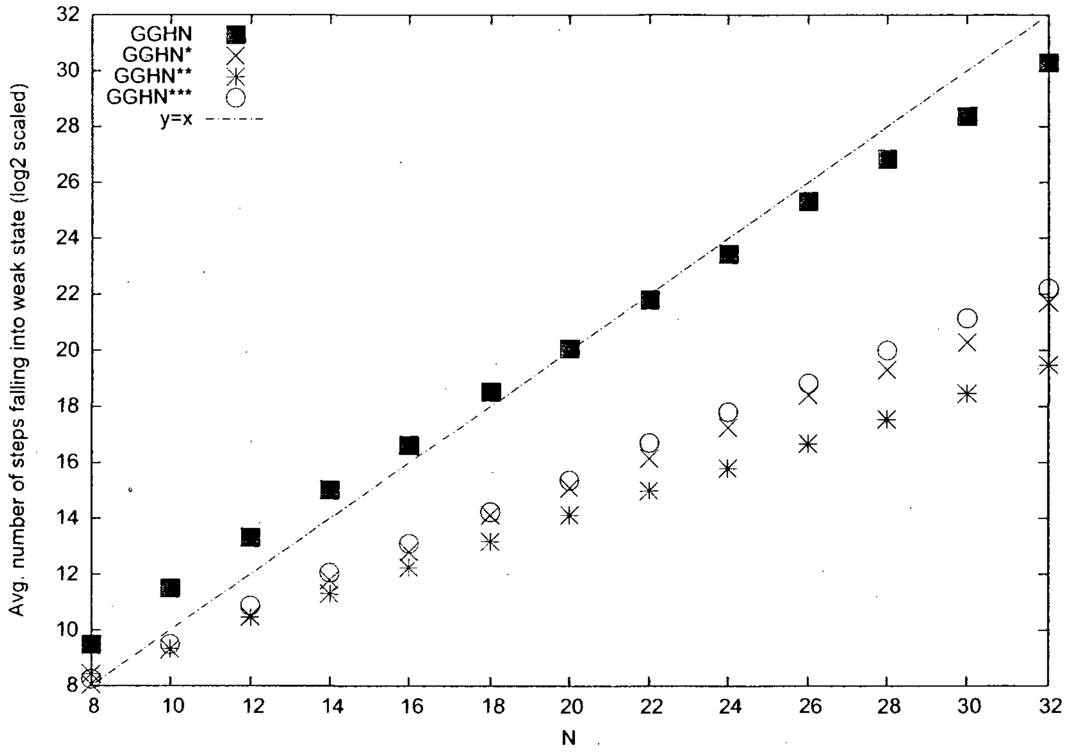

Figure 5: Experimental results for different variants of GGHN(n,32): average number of steps needed for entering the weak state versus $l(N) = N$ line

5.7 Experimental results on absorption time of different GGHN variants

In this section we provide some experimental results for the absorbtion time of different GGHN(n,32) variants. Let

- GGHN denote the original GGHN(n,32)
- GGHN* denote the idealized model described in the previous section
- GGHN** denote the modified GGHN cipher where an S table swap operation, $Swap(S[i], S[j])$) (originally used in both RC4 and NGG), is inserted before the output step in the

GGHN PRGA.

- GGHN*** denote the modified GGHN cipher where the least significant bit of t is XORed with a binary random variable right after the $t = S[i] + S[j] \bmod N$ step.

The modifications in the code of GGHN** and GGHN*** are done in order to eliminate the bias in the t value towards becoming even whenever the S table tends to have a large number of even entries. Furthermore, we extend the definition of GGHN(n,m) which enforces power-of-two table sizes to GGHN variants in which the S table size, N, can be any even number.

Figure 5 shows the base 2 logarithm of the average value of the absorbtion time. Let $2^{l(N)}$ denote the average absorbtion time of the cipher with $m = 32$. Using the minimum least square error approximation, then, from the experimental data, $l(N)$ for the above ciphers can be approximated by

$$l(N) \approx \begin{cases} 0.854 \times N + 2.959 \text{ for GGHN} \\ 0.557 \times N + 3.846 \text{ for GGHN*} \\ 0.454 \times N + 4.920 \text{ for GGHN**} \\ 0.579 \times N + 3.811 \text{ for GGHN***} \end{cases}$$

It is clear that, as N increases, in all cases $l(N)$ is less than N. Substituting $N = 256$ into the $l(N)$ formula above indicates that the number of steps needed before GGHN enters weak state is about 2^{222} steps, instead of 2^{256} steps. The above estimates should be interpreted with care as it might be the case that GGHN does not behaves this way for $N = 256$.

Providing a theoretically sound argument for this claim seems to be difficult due to the fact that GGHN does not satisfy the Markovian assumption.

It should also be noted that unlike variants of GGHN, it is not guaranteed for the original GGHN to enter the weak state. Instead, it can enter a cycle with respect to least significant bit. This kind of cycles occurs if all LSB internal state values at different $i = 0$ moments (close one to another) are equal and this repeats to infinity. Analysis of this type of behavior is out of the scope of this chapter, but we noted that the probability that this will happen drops dramatically when N increases. According to experimental results, for $N = 16$, this kind of cycle happens for around 5% of GGHN instantiations. For $N \geq 32$, throughout our experiments, we did not observe any cycles and the GGHN always reaches the weak state.

Chapter 6

Conclusion and future work

In this thesis, we provided analysis of two block ciphers and two stream ciphers. In particular we have shown that:

- The key scheduling of the TREYFER block cipher seems to be oversimplified and that yields several weaknesses. We presented a related-key attack requiring only 2^{11} chosen plaintext encryptions. While it might be argued that it is unlikely that an attacker can persuade a human operator to encrypt plaintexts under the 8 related keys, modern cryptography is implemented using complex protocols, and in some cases a related-key attack can be made feasible. In these scenarios, our attack can recover the secret keys in few milliseconds.

- PIFEA-M algorithm is still vulnerable to a the differential style attack that it was supposedly designed to resist. Using only four plaintext ciphertext pairs, and by solving a set of linear equations over $GF(2)$, all successive ciphertext can be deciphered even without explicitly recovering the secret key. On the other hand, one should note that, while the given cryptanalysis is based on an explicit assumption from PIFEA-M algorithm specifications, the considered attack scenario is not a usual one.

- The NGG key scheduling algorithm does not randomize its initial internal state table sufficiently. In particular, after the KSA, the cipher internal state is highly biased which allows us to distinguish NGG from a random stream based only on the first keystream word. Furthermore, the bias left by the KSA reveals information about the secret key. Since NGG PRGA sends the internal state elements to output without masking, information about the secret key can be recovered by examining the first few kilobytes of the keystream. Our experimental results show that for approximately $2^{119.2}$ of the 128-bit keys, the secret key can be recovered with around 2^{32} steps. Based on the analysis above, it is clear that NGG is insecure. Designing an efficient and secure generalization of RC4 for 32/64-bit processors remains a challenging research problem.

- Many variants of GGHN-like ciphers with table size N may need substantially less than 2^N steps to enter a weak state. This was done by modelling an idealized GGHN(n,m) by a Markov chain, and calculating chain's absorption time. We have also shown that, if the designers of GGHN kept the swap operation originally used by both RC4 and NGG for the S table scrambling during the pseudorandom generation algorithm, then the cipher would enter the weak state much faster than expected when treating the S table, during the PRGA update process, as a purely random table.

To extend these results, further research can be conducted in the following directions:

- For TREYFER, it is interesting to investigate if there are other permutations of plaintexts and keys that derive similar ciphertexts. It is also interesting to provide further analysis of TREYFER which employs an enhanced key scheduling algorithm (e.g., by adding some round dependent constants).

- Since the original FEA-M cipher is inherently linear, it would be interesting to put more effort in the cryptanalysis of other FEA-M variants and test it with respect to other known cryptanalytic attacks listed in Chapter 2.

- Since the amount of information about the secret key left by the NGG KSA procedure grows with increasing the word size from 32 to 64, it would be interesting to derive a full key recovery algorithm which succeeds with high probability when the word size is 64.

- The question of how to exactly calculate the number of steps required for GGHN to enter a weak state is open. A new method for calculating absorption times of non-Markovian chains is required, for the case when the computational complexity of experimentally finding these values is too high.

- The design of light weight, yet secure, ciphers suitable for resource constrained environments still presents a challenging goal for cryptographers.

Bibliography

[1] H. Ahmadi, T. Eghlidos, and S. Khazaei, *Improved Guess and Determine Attack on SOSEMANUK*, eSTREAM report 2005/085 (2005). Avilable at http://www.ecrypt.eu.org/stream/papers.html

[2] E. R. Berlekamp, *Algebraic coding theory*, McGraw-Hill, New York, 1968.

[3] D. Bernstein, *Which eStream cipher have been broken?*, ECRYPT report, 2008, Available at http://www.ecrypt.eu.org/stream/papersdir/2008/010.pdf.

[4] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, H. Sibert, *Sosemanuk: a fast software-oriented stream cipher*, eSTREAM report 2005/027 (2005). Avilable at http://www.ecrypt.eu.org/stream/papers.html

[5] E. Biham and Y. Carmeli, *Efficient Reconstruction of RC4 Keys from Internal States*, Proc. of Fast Software Encryption, FSE 2008, pp. 270-288, LNCS 5086, Springer-Verlag, 2008.

[6] E.Biham, A.Biryukov, A.Shamir, *Cryptanalysis of Skipjack Reduced to 31 Rounds using Impossible Differentials*, In J. Stern, editor, Advances in Cryptology: EUROCRYPT'99, LNCS 1592, pp. 12-23. Springer Verlag, 1999.

[7] E. Biham, *New Types of Cryptanalytic Attacks Using Related Keys*, Advances in Cryptology, EUROCRYPT '93, Springer-Verlag, 1994, pp. 398-409.

[8] E.Biham, A.Shamir, *Differential Cryptanalysis of the Full 16-Round DES*, LNCS-740, Proceedings of Crypto'92, Springer-Verlag, 1992.

[9] E. Biham, O. Dunkelman and N. Keller, *Related-Key Impossible Differential Attacks on 8-Round AES-192*, Topics in Cryptolog: CT-RSA 2006, LNCS 3860, 2006, pp. 21-33.

[10] E. Biham, O. Dunkelman and N. Keller, *A Related-Key Rectangle Attack on the Full KASUMI*, Proc. of ASIACRYPT 2005, LNCS 3788, 2005, pp. 443-461.

[11] Alex Biryukov and David Wagner, *Slide Attacks*, Proc, of the 6th International Workshop on Fast Software Encryption (FSE '99), pp. 245-259, Rome: Springer-Verlag.

[12] A. Biryukov and A. Shamir, *Structural Cryptanalysis of SASAS*, Proceedings of Eurocrypt'01, pp 394-405, Lecture Notes in Computer Science, volume 2045, Springer-Verlag

[13] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen and O. Scavenius, *Rabbit: A new high-performance Stream Cipher*, Proc. of FSE 2003, Springer-Verlag, LNCS 2887, pp. 307-329.

[14] 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*, CHES 2007, LNCS4727, pp.450- 466, Springer, 2007.

[15] A. G. Chefranov, *Parametrized Improved Fast Encryption Algorithm for Multimedia PIFEA-M*, IEEE Comm. Letters, Vol. 12, No. 6, pp. 404-406, June 2008.

[16] J. Daemen, L. Knudsen and V. Rijmen, *The Block Cipher SQUARE*, Fast Software Encryption 1997, Lecture Notes in Computer Science Volume 1267, pp 149-165, Springer-Verlag.

[17] Joan Daemen and Vincent Rijmen, *AES proposal: Rijndael. In AES Round 1 Technical Evaluation*, CD-1: Documentation. NIST, August 1998. URL: http://www.esat.kuleuven.ac.be/ rijmen/rijndael/ or http://www.nist.gov/aes.

[18] S.R. Fluhrer and D.A. McGrew, *Statistical Analysis of the Alleged RC4 Keystream Generator*, Proc. of Fast Software Encryption, FSE 2000, pp. 19-30, LNCS 1978, Springer-Verlag, 2000.

[19] S.R. Fluhrer, I. Mantin and A. Shamir, *Weaknesses in the Key Scheduling Algorithm of RC4*, Proc. of Selected Areas in Cryptography, SAC 2001, pp. 1-24, LNCS2259, Springer-Verlag, 2001.

[20] Geffe, P. R., *How to protect data with ciphers that are really hard to break*, Electronics, Jan. 1973, pg. 99-101

[21] C. G. Gunther, *Alternating step generators controlled by de Bruijn sequences*, Advances in Cryptology - EuroCrypt '87 (p5-14), LNCS 304, Springer Verlag, ISBN 3-540-19102-X

[22] J.D. Golić, Linear Statistical Weakness of Alleged RC4 Keystream Generator, LNCS-1233, EUROCRYPT '97, pp. 226-238, Springer-Verlag, 1997.

[23] G. Gong, K. Chand, M. Hell and Y. Nawaz,*Towards a General RC4-Like Keystream Generator*, Proc. of CISC 2005, pp. 162-174, LNCS 3822, Springer-Verlag, 2005.

[24] Charles M. Grinstead, J. Laurie Snell, Introduction to Probability, American Mathematical Society, 2nd edition, 1997

[25] P. Hawkes and G. G. Rose, *On the applicability of distinguishing attacks against stream ciphers*, Proc. of the Third NESSIE Workshop, 2002.

[26] M.E.Hellman and S.K.Langford, *Differential-linear cryptanalysis*, Advances in Cryptology - Proc. Crypto'94, LNCS 839, pages 26Ã¢39. Springer Verlag, 1994.

[27] Thomas Jakobsen and Lars R. Knudsen, *The Interpolation Attack on Block Ciphers*,in the proceedings of Fast Software Encryption (FSE '97), Lecture Notes in Computer Science Volume 1267, pages 28-40, Springer-Verlag.

[28] J. Kelsey, B. Schneier and D. Wagner, *Related-key cryptanalysis of 3-WAY, Biham-DES,CAST, DES-X, NewDES, RC2, and TEA*, Proc. of Information and Communications Security, LNCS 1334, 1997, pp. 233-246.

[29] A. Klein. Attacks on the RC4 stream cipher. Personal Andreas Klein website (2006) http://cage.ugent.be/ klein/RC4/RC4-en.ps

[30] L.Knudsen, *Truncated and Higher Order Differentials*, Proceedings of the Second International Workshop on Fast Software Encryption, Leuven, Belgium, LNCS 1008, Springer, pp.196-211, 1995

[31] L.R.Knudsen, M.J.B.Robshaw and D.Wagner, *Truncated Differentials and Skipjack*, CRYPTO 1999, pp. 165-180, Springer-Verlag, 1999.

[32] L.R.Knudsen, *Partial and higher order differentials and its application to the DES*, BRICS report series, RS-95-9, ISSN 0909-0878, February 1995.

[33] L.R. Knudsen, W. Meier, B. Prenel, V. Rijmen and S. Verdoolaege, *Analysis Methods for (Alleged) RC4*, Proc. of ASIACRYPT'98, pp. 327-341, LNCS 1514, Springer-Verlag, 1998.

[34] S. Li and K. T. Lo, *Security Problems with Improper Implementations of Improved FEA-M* Journal of Systems & Software, Vol. 80, No. 5, pp. 791-794, 2007.

[35] S. Lucks, *Ciphers Secure against Related-Key Attacks*, Proc. of Fast Software Encryption, LNCS 3017, 2004, pp. 359-370.

[36] I. Mantin and A. Shamir,*A practical Attack on Broadcast RC4*, Proc. of Fast Software Encryption, FSE 2001, pp. 152-164, LNCS2355, Springer-Verlag, 2001.

[37] I. Mantin, *Predicting and Distinguishing Attacks on RC4 Keystream Generator*, Proc. of EUROCRYPT' 2005, pp. 491-506, LNCS 3494 Springer-Verlag, 2005.

[38] I. Mantin, *A Practical Attack on the Fixed RC4 in the WEP Mode*, Proc. of ASIACRYPT 2005, pp.395-411, LNCS 3788, Springer-Verlag, 2005.

[39] James L. Massey, *SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm*, Proc. of the first workshop on Fast Software Encryption (FSE'93), pp. 1-17, Springer-Verlag.

[40] J. L. Massey, *Shift-register synthesis and BCH decoding*, IEEE Trans. on Inform. Theory vol. IT-15, January 1969

[41] M. Matsui, *Linear Cryptanalysis Method for DES Cipher*, LNCS, Advances in Cryptology, proceedings of EUROCRYPT'93, pp. 386-397, 1993.

[42] A. Maximov, *Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers*, Proc. of Fast Software Encryption, FSE 2005, LNCS 3357, pp. 342-358. Springer-Verlag, 2005.

[43] A. Maximov and D. Khovratovich, *New State Recovery Attack on RC4*, Proc. of CRYPTO 2008, pp. 297-316, LNCS 5157, Springer-Verlag, 2008.

[44] A J. Menezes, P. C. van Oorschot and S A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[45] M. J. Mihaljevic and R. Kohno, *Cryptanalysis of fast encryption algorithm for multimedia FEA-M*, IEEE Communications Letters, Vol 6, Issue 9, pp. 382-384, 2002

[46] M. J. Mihaljevic, *On Vulnerabilities and Improvements of Fast Encryption algorithm for Multimedia FEA-M*, IEEE Trans. on Consumer Electronics, Vol. 49, No. 4, November 2003.

[47] I. Mironov, Not (So) Random Shuffle RC4, Crypto 2002, LNCS-2442, pp. 304-319, Springer-Verlag, 2002

[48] S. Mister and S. Tavares, *Cryptanalysis of RC4-like Ciphers*, Proc. of Selected Areas in Cryptography, SAC 98, pp. 131-143, LNCS 1556, Springer-Verlag, 1999.

[49] National Institute of Standards and Technology, *FIPS-197: Advanced Encryption Standard*, November 2001.

[50] Y. Nawaz, K.C. Gupta and G. Gong, *A 32-bit RC4-like keystream generator*, Technical Report CACR 2005-21, Center for Applied Cryptographic Research, University of Waterloo, 2005. Also available at Cryptology ePrint Archive, 2005-175, `http://eprint.iacr.org/2005/175`

[51] S. Paul, B. Preenel, *On the (In)security of Stream Ciphers Based on Arrays and Modular Addition*, Proc. of ASIACRYPT 2006, pp. 69-83, LNCS 4284, Springer-Verlag, 2006.

[52] S. Paul, B. Preenel, *A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher*, Proc. of Fast Software Encryption, FSE 2004, LNCS 3017, pp. 245ÂđÂű259. Springer-Verlag, 2004.

[53] G. Paul and S. Maitra, *Permutation After RC4 Key Scheduling Reveals the Secret Key*, Proc. of Selected Areas in Cryptography, SAC 2007, pp. 360-377, LNCS 4876, Springer-Verlag, 2007.

[54] S. Paul and B. Preneel, *A New Weakness in RC4 Keystream Generator and an Approach to Improve the Security of the Cipher*, Proc. of Fast Software Encryption, FSE 2004, pp 245-259, LNCS 3017, Springer-Verlag, 2004.

[55] S. Paul and B. Preneel, *Analysis of Non-fortuitous Predictive States of the RC4 Keystream Generator*, Proc. of Indocrypt 2003, pp. 52-67, LNCS 2904, Springer-Verlag, 2003.

[56] B. Schneier, *Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish)*, Fast Software Encryption 1993: 191-204, 1993

[57] G. Sekar, S. Paul and B. Preneel, *Related-Key Attacks on the Py-Family of Ciphers and an Approach to Repair the Weaknesses*, Proc. of INDOCRYPT 2007, LNCS 4859, 2007, pp. 58-72.

[58] T. Siegenthaler, *Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications*, IEEE Transactions on Information Theory 30 (5), pp. 776-780, 1984

[59] E. Tews, R.P. Weinmann and A. Pyshkin, Breaking of 104 Bit WEP in Less than 60 Seconds, 2007, Cryptology ePrint Archive, 2007-120, IACR 2007, Available at http://eprint.iacr.org/2007/120

[60] V. Tomašević , S. Bojanić, O. Nieto-Taladriz, *Finding an internal state of RC4 stream cipher*, Information Sciences: an International Journal, Vol 177; No. 7, pp. 1715-1727, 2007.

[61] Y. Tsunoo, T. Saito, H. Kubo, and T. Suzaki, *A Distinguishing Attack on a Fast Software-Implemented RC4-Like Stream Cipher*, IEEE Trans. on Information Theory, Vol. 53, No. 9, 2007.

[62] S. Vaudenay and M. Vuagnoux, *Passive-Only Key Recovery Attacks on RC4*, Proc. of Selected Areas in Cryptography, SAC 2007, pp. 344-359, LNCS 4876, Springer-Verlag, 2007.

[63] M. Wang, *Differential Cryptanalysis of PRESENT*, In proc. of Africacrypt 2008, to appear, Also available at Cryptology ePrint Archive: Report 2007/408.

[64] D.Wagner, *The Boomerang Attack*, in the Proceedings of FSE999, LNCS-1636, Springer-Verlag.

[65] D. J. Wheeler and R. M. Needham, *TEA, a tiny encryption algorithm*, Proc. of the 2nd workshop on Fast Software Encryption (FSE'94), pp. 363-366, Leuven, Belgium, 1994, Springer-Verlag.

[66] H. Wu, *Cryptanalysis of a 32-bit RC4-like Stream Cipher*, Cryptology ePrint Archive, 2005-219, IACR 2005, Available at eprint.iacr.org/2005/219.pdf

[67] X. Yi, C. K. Tan, C. K. Siew and M. R. Syed, *Fast Encryption for Multimedia*, IEEE Trans. Consumer Electronics, Vol. 47, No. 1, pp. 101-107, Feb. 2001.

[68] X. Yi, C. K. Tan, C. K. Siew and M. R. Syed, *ID-based key agreement for multimedia encryption*, IEEE Trans. Consumer Electronics, Vol. 48, No. 2, pp. 298-303, May 2002

[69] A. M. Youssef and S. E. Tavares, *Comments on the Security of Fast Encryption Algorithm for Multimedia (FEA-M)*, IEEE Trans. on Consumer Electronics, Vol. 49, No. 1, Feb 2003.

[70] G. Yuval, *Reinventing the Travois: Encryption/MAC in 30 ROM Bytes*, Proc. of the 4th International Workshop on Fast Software Encryption (FSE '97), pp. 205-209, Springer-Verlag, 1997.

[71] W. Zhang, L. Zhang, W. Wu and D. Feng, *Related-Key Differential-Linear Attacks on Reduced AES-192*, Proc. of INDOCRYPT 2007, LNCS 4859, pp. 73-85.

[72] B. Zoltak, *VMPC one-way function and stream cipher*, Proc. of Fast Software Encryption, FSE 2004, LNCS 3017, pp. 210-225, Springer-Verlag, 2004.