Lecture Notes in Computer Science 1978
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Bruce Schneier (Ed.)

Fast Software Encryption

7th International Workshop, FSE 2000
New York, NY, USA, April 10-12, 2000
Proceedings

Springer

Volume Editor

Bruce Schneier
Counterpane Internet Security, Inc.
3031 Tisch Way, Suite 100PE, San Jose, CA 95128, USA
E-mail: schneier@counterpane.com

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Fast software encryption : 7th international workshop ; proceedings /
FSE 2000, New York, NY, April 10 - 12, 2000. Bruce Schneider (ed.). -
Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ;
Milan ; Paris ; Singapore ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 1978)
ISBN 3-540-41728-1

CR Subject Classification (1998): E.3, F.2.1, E.4, G.4

ISSN 0302-9743
ISBN 3-540-41728-1 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP Berlin, Stefan Sossna
Printed on acid-free paper SPIN 10781399 06/3142 5 4 3 2 1 0

Preface

Since 1993, cryptographic algorithm research has centered around the Fast Software Encryption (FSE) workshop. First held at Cambridge University with 30 attendees, it has grown over the years and has achieved worldwide recognition as a premiere conference. It has been held in Belgium, Israel, France, Italy, and, most recently, New York.

FSE 2000 was the 7th international workshop, held in the United States for the first time. Two hundred attendees gathered at the Hilton New York on Sixth Avenue, to hear 21 papers presented over the course of three days: 10–12 April 2000. These proceedings constitute a collection of the papers presented during those days.

FSE concerns itself with research on classical encryption algorithms and related primitives, such as hash functions. This branch of cryptography has never been more in the public eye. Since 1997, NIST has been shepherding the Advanced Encryption Standard (AES) process, trying to select a replacement algorithm for DES. The first AES conference, held in California the week before Crypto 98, had over 250 attendees. The second conference, held in Rome two days before FSE 99, had just under 200 attendees. The third AES conference was held in conjunction with FSE 2000, during the two days following it, at the same hotel.

It was a great pleasure for me to organize and chair FSE 2000. We received 53 submissions covering the broad spectrum of classical encryption research. Each of those submissions was read by at least three committee members – more in some cases. The committee chose 21 papers to be presented at the workshop. Those papers were distributed to workshop attendees in a preproceedings volume. After the workshop, authors were encouraged to further improve their papers based on comments received. The final result is the proceedings volume you hold in your hand.

To conclude, I would like to thank all the authors who submitted papers to this conference, whether or not your papers were accepted. It is your continued research that makes this field a vibrant and interesting one. I would like to thank the other program committee members: Ross Anderson (Cambridge), Eli Biham (Technion), Don Coppersmith (IBM), Cunsheng Ding (Singapore), Dieter Gollmann (Microsoft), Lars Knudsen (Bergen), James Massey (Lund), Mitsuru Matsui (Mitsubishi), Bart Preneel (K.U.Leuven), and Serge Vaudenay (EPFL). They performed the hard – and too often thankless – task of selecting the program. I'd like to thank my assistant, Beth Friedman, who handled administrative matters for the conference. And I would like to thank the attendees for coming to listen, learn, share ideas, and participate in the community. I believe that FSE represents the most interesting subgenre within cryptography, and that this conference represents the best of what cryptography has to offer.

Enjoy the proceeedings, and I'll see everyone next year in Japan.

August 2000 Bruce Schneier

Table of Contents

Real Time Cryptanalysis of A5/1 on a PC

Alex Biryukov[1], Adi Shamir[1], and David Wagner[2]

[1] Computer Science department, The Weizmann Institute, Rehovot 76100, Israel
[2] Computer Science department, University of California, Berkeley CA 94720, USA.

Abstract. A5/1 is the strong version of the encryption algorithm used by about 130 million GSM customers in Europe to protect the over-the-air privacy of their cellular voice and data communication. The best published attacks against it require between 2^{40} and 2^{45} steps. This level of security makes it vulnerable to hardware-based attacks by large organizations, but not to software-based attacks on multiple targets by hackers.

In this paper we describe new attacks on A5/1, which are based on subtle flaws in the tap structure of the registers, their noninvertible clocking mechanism, and their frequent resets. After a 2^{48} parallelizable data preparation stage (which has to be carried out only once), the actual attacks can be carried out in real time on a single PC.

The first attack requires the output of the A5/1 algorithm during the first two minutes of the conversation, and computes the key in about one second. The second attack requires the output of the A5/1 algorithm during about two seconds of the conversation, and computes the key in several minutes. The two attacks are related, but use different types of time-memory tradeoffs. The attacks were verified with actual implementations, except for the preprocessing stage which was extensively sampled rather than completely executed.

REMARK: We based our attack on the version of the algorithm which was derived by reverse engineering an actual GSM telephone and published at `http://www.scard.org`. We would like to thank the GSM organization for graciously confirming to us the correctness of this unofficial description. In addition, we would like to stress that this paper considers the narrow issue of the cryptographic strength of A5/1, and not the broader issue of the practical security of fielded GSM systems, about which we make no claims.

1 Introduction

The over-the-air privacy of GSM telephone conversations is protected by the A5 stream cipher. This algorithm has two main variants: The stronger A5/1 version is used by about 130 million customers in Europe, while the weaker A5/2 version is used by another 100 million customers in other markets. The approximate design of A5/1 was leaked in 1994, and the exact design of both A5/1 and A5/2 was reverse engineered by Briceno from an actual GSM telephone in 1999 (see [3]).

In this paper we develop two new cryptanalytic attacks on A5/1, in which a single PC can extract the conversation key in real time from a small amount of generated output. The attacks are related, but each one of them optimizes a different parameter: The first attack (called **the biased birthday attack**) requires two minutes of data and one second of processing time, whereas the second attack (called the **the random subgraph attack**) requires two seconds of data and several minutes of processing time. There are many possible choices of tradeoff parameters in these attacks, and three of them are summarized in Table 1.

Table 1. Three possible tradeoff points in the attacks on A5/1.

Attack Type	Preprocessing steps	Available data	Number of 73GB disks	Attack time
Biased Birthday attack (1)	2^{42}	2 minutes	4	1 second
Biased Birthday attack (2)	2^{48}	2 minutes	2	1 second
Random Subgraph attack	2^{48}	2 seconds	4	minutes

Many of the ideas in these two new attacks are applicable to other stream ciphers as well, and define new quantifiable measures of security.

The paper is organized in the following way: Section 2 contains a full description of the A5/1 algorithm. Previous attacks on A5/1 are surveyed in Section 3, and an informal description of the new attacks is contained in Section 4. Finally, Section 5 contains various implementation details and an analysis of the expected success rate of the attacks, based on large scale sampling with actual implementations.

2 Description of the A5/1 Stream Cipher

A GSM conversation is sent as a sequence of frames every 4.6 millisecond. Each frame contains 114 bits representing the digitized A to B communication, and 114 bits representing the digitized B to A communication. Each conversation can be encrypted by a new session key K. For each frame, K is mixed with a publicly known frame counter F_n, and the result serves as the initial state of a generator which produces 228 pseudo random bits. These bits are XOR'ed by the two parties with the 114+114 bits of the plaintext to produce the 114+114 bits of the ciphertext.

A5/1 is built from three short linear feedback shift registers (LFSR) of lengths 19, 22, and 23 bits, which are denoted by $R1, R2$ and $R3$ respectively. The rightmost bit in each register is labelled as bit zero. The taps of $R1$ are at bit positions 13,16,17,18; the taps of $R2$ are at bit positions 20,21; and the taps of $R3$ are at bit positions 7, 20,21,22 (see Figure 1). When a register is clocked, its taps are XORed together, and the result is stored in the rightmost bit of the left-shifted register. The three registers are maximal length LFSR's with periods $2^{19}-1$, $2^{22}-1$, and $2^{23}-1$, respectively. They are clocked in a stop/go fashion using the following majority rule: Each register has a single "clocking" tap (bit 8 for $R1$, bit 10 for $R2$, and bit 10 for for $R3$); each clock cycle, the majority function of the clocking taps is calculated and only those registers whose clocking taps agree with the majority bit are actually clocked. Note that at each step either two or three registers are clocked, and that each register moves with probability 3/4 and stops with probability 1/4.

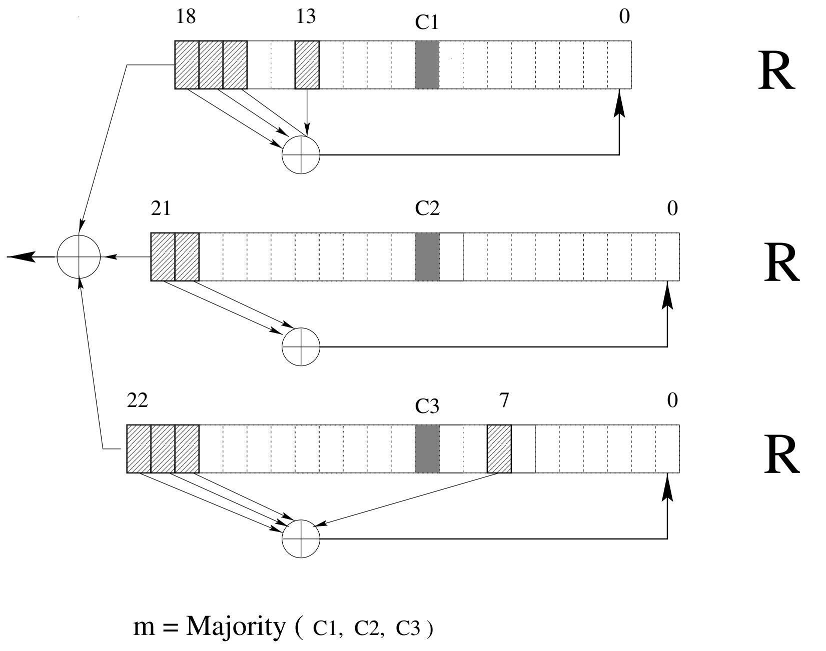


Fig. 1. The A5/1 stream cipher.

The process of generating pseudo random bits from the session key K and the frame counter F_n is carried out in four steps:

– The three registers are zeroed, and then clocked for 64 cycles (ignoring the stop/go clock control). During this period each bit of K (from lsb to msb) is XOR'ed in parallel into the lsb's of the three registers.
– The three registers are clocked for 22 additional cycles (ignoring the stop/go clock control). During this period the successive bits of F_n (from lsb to msb) are again XOR'ed in parallel into the lsb's of the three registers. The contents of the three registers at the end of this step is called the **initial state** of the frame.

– The three registers are clocked for 100 additional clock cycles with the stop/go clock control but without producing any outputs.
– The three registers are clocked for 228 additional clock cycles with the stop/go clock control in order to produce the 228 output bits. At each clock cycle, one output bit is produced as the XOR of the msb's of the three registers.

3 Previous Attacks

The attacker is assumed to know some pseudo random bits generated by A5/1 in some of the frames. This is the standard assumption in the cryptanalysis of stream ciphers, and we do not consider in this paper the crucial issue of how one can obtain these bits in fielded GSM systems. For the sake of simplicity, we assume that the attacker has complete knowledge of the outputs of the A5/1 algorithm during some initial period of the conversation, and his goal is to find the key in order to decrypt the remaining part of the conversation. Since GSM telephones send a new frame every 4.6 milliseconds, each second of the conversation contains about 2^8 frames.

At the rump session of Crypto 99, Ian Goldberg and David Wagner announced an attack on A5/2 which requires very few pseudo random bits and just $O(2^{16})$ steps. This demonstrated that the "export version" A5/2 is totally insecure.

The security of the A5/1 encryption algorithm was analyzed in several papers. Some of them are based on the early imprecise description of this algorithm, and thus their details have to be slightly modified. The known attacks can be summarized in the following way:

– Briceno[3] found out that in all the deployed versions of the A5/1 algorithm, the 10 least significant of the 64 key bits were always set to zero. The complexity of exhaustive search is thus reduced to $O(2^{54})$. [1]
– Anderson and Roe[1] proposed an attack based on guessing the 41 bits in the shorter R_1 and R_2 registers, and deriving the 23 bits of the longer R_3 register from the output. However, they occasionally have to guess additional bits to determine the majority-based clocking sequence, and thus the total complexity of the attack is about $O(2^{45})$. Assuming that a standard PC can test ten million guesses per second, this attack needs more than a month to find one key.
– Golic[4] described an improved attack which requires $O(2^{40})$ steps. However, each operation in this attack is much more complicated, since it is based on the solution of a system of linear equations. In practice, this algorithm is not likely to be faster than the previous attack on a PC.

[1] Our new attack is not based on this assumption, and is thus applicable to A5/1 implementations with full 64 bit keys. It is an interesting open problem whether we can speed it up by assuming that 10 key bits are zero.

– Golic[4] describes a general time-memory tradeoff attack on stream ciphers (which was independently discovered by Babbage [2] two years earlier), and concludes that it is possible to find the A5/1 key in 2^{22} probes into random locations in a precomputed table with 2^{42} 128 bit entries. Since such a table requires a 64 terabyte hard disk, the space requirement is unrealistic. Alternatively, it is possible to reduce the space requirement to 862 gigabytes, but then the number of probes increases to $O(2^{28})$. Since random access to the fastest commercially available PC disks requires about 6 milliseconds, the total probing time is almost three weeks. In addition, this tradeoff point can only be used to attack GSM phone conversations which last more than 3 hours, which again makes it unrealistic.

4 Informal Description of the New Attacks

We start with an executive summary of the key ideas of the two attacks. More technical descriptions of the various steps will be provided in the next section.

Key idea 1: Use the Golic time-memory tradeoff. The starting point for the new attacks is the time-memory tradeoff described in Golic[3], which is applicable to any cryptosystem with a relatively small number of internal states. A5/1 has this weakness, since it has $n = 2^{64}$ states defined by the $19+22+23 = 64$ bits in its three shift registers. The basic idea of the Golic time-memory tradeoff is to keep a large set A of precomputed states on a hard disk, and to consider the large set B of states through which the algorithm progresses during the actual generation of output bits. Any intersection between A and B will enable us to identify an actual state of the algorithm from stored information.

Key idea 2: Identify states by prefixes of their output sequences. Each state defines an infinite sequence of output bits produced when we start clocking the algorithm from that state. In the other direction, states are usually uniquely defined by the first $log(n)$ bits in their output sequences, and thus we can look for equality between unknown states by comparing such prefixes of their output sequences. During precomputation, pick a subset A of states, compute their output prefixes, and store the (prefix, state) pairs sorted into increasing prefix values. Given actual outputs of the A5/1 algorithm, extract all their (partially overlapping) prefixes, and define B as the set of their corresponding (unknown) states. Searching for common states in A and B can be efficiently done by probing the sorted data A on the hard disk with prefix queries from B.

Key idea 3: A5/1 can be efficiently inverted. As observed by Golic, the state transition function of A5/1 is not uniquely invertible: The majority clock control rule implies that up to 4 states can converge to a common state in one clock cycle, and some states have no predecessors. We can run A5/1 backwards by exploring the tree of possible predecessor states, and backtracking from dead ends. The average number of predecessors of each node is 1, and thus the expected number of vertices in the first k levels of each tree grows only linearly in k (see[3]). As a result, if we find a common state in the disk and data, we can obtain a small number of candidates for the initial state of the frame. The weakness we exploit here is that due to the frequent reinitializations there is a very short distance from intermediate states to initial states.

Key idea 4: The key can be extracted from the initial state of any frame. Here we exploit the weakness of the A5/1 key setup routine. Assume that we know the state of A5/1 immediately after the key and frame counter were used, and before the 100 mixing steps. By running backwards, we can eliminate the effect of the known frame counter in a unique way, and obtain 64 linear combinations of the 64 key bits. Since the tree exploration may suggest several keys, we can choose the correct one by mixing it with the next frame counter, running A5/1 forward for more than 100 steps, and comparing the results with the actual data in the next frame.

Key idea 5: The Golic attack on A5/1 is marginally impractical. By the well known birthday paradox, A and B are likely to have a common state when their sizes a and b satisfy $a * b \approx n$. We would like a to be bounded by the size of commercially available PC hard disks, and b to be bounded by the number of overlapping prefixes in a typical GSM telephone conversation. Reasonable bounds on these values (justified later in this paper) are $a \approx 2^{35}$ and $b \approx 2^{22}$. Their product is 2^{57}, which is about 100 times smaller than $n = 2^{64}$. To make the intersection likely, we either have to increase the storage requirement from 150 gigabytes to 15 terabytes, or to increase the length of the conversation from two minutes to three hours. Neither approach seems to be practical, but the gap is not huge and a relatively modest improvement by two orders of magnitude is all we need to make it practical.

Key idea 6: Use special states. An important consideration in implementing time-memory tradeoff attacks is that access to disk is about a million times slower than a computational step, and thus it is crucial to minimize the number of times we look for data on the hard disk. An old idea due to Ron Rivest is to keep on the disk only special states which are guaranteed to produce output bits starting with a particular pattern α of length k, and to access the disk only when we encounter such a prefix in the data. This reduces the number b of disk probes by a factor of about 2^k. The number of points a we have to memorize remains unchanged, since in the formula $a * b \approx n$ both b and n are reduced by the same factor 2^k. The downside is that we have to work 2^k times harder during the preprocessing stage, since only 2^{-k} of the random states we try produce outputs with such a k bit prefix. If we try to reduce the number of disk access steps in the time memory attack on A5/1 from 2^{22} to 2^6, we have to increase the preprocessing time by a factor of about 64,000, which makes it impractically long.

Key idea 7: Special states can be efficiently sampled in A5/1. A major weakness of A5/1 which we exploit in both attacks is that it is easy to generate all the states which produce output sequences that start with a particular k-bit pattern α with $k = 16$ without trying and discarding other states. This is due to a poor choice of the clocking taps, which makes the register bits that affect the clock control and the register bits that affect the output unrelated for about 16 clock cycles, so we can choose them independently. This easy access to special states does not happen in good block ciphers, but can happen in stream ciphers due to their simpler transition functions. In fact, the maximal value of k for which special states can be sampled without trial and error can serve as a new security measure for stream ciphers, which we call its **sampling resistance**. As demonstrated in this paper, high values of k can have a big impact on the efficiency of time-memory tradeoff attacks on such cryptosystems.

Key idea 8: Use biased birthday attacks. The main idea of the first attack is to consider sets A and B which are not chosen with uniform probability distribution among all the possible states. Assume that each state s is chosen for A with probability $P_A(s)$, and is chosen for B with probability $P_B(s)$. If the means of these probability distributions are a/n and b/n, respectively, then the expected size of A is a, and the expected size of B is b.

The birthday threshold happens when $\sum_s P_A(s)P_B(s) \approx 1$. For independent uniform distributions, this evaluates to the standard condition $a*b \approx n$. However, in the new attack we choose states for the disk and states in the data with two non-uniform probability distributions which have strong positive correlation. This makes our time memory tradeoff much more efficient than the one used by Golic. This is made possible by the fact that in A5/1, the initial state of each new frame is rerandomized very frequently with different frame counters.

Key idea 9: Use Hellman's time-memory tradeoff on a subgraph of special states. The main idea of the second attack (called the random subgraph attack) is to make most of the special states accessible by simple computations from the subset of special states which are actually stored in the hard disk. The first occurrence of a special state in the data is likely to happen in the first two seconds of the conversation, and this single occurrence suffices in order to locate a related special state in the disk even though we are well below the threshold of either the normal or the biased birthday attack. The attack is based on a new function f which maps one special state into another special state in an easily computable way. This f can be viewed as a random function over the subspace of 2^{48} special states, and thus we can use Hellman's time-memory tradeoff[4] in order to invert it efficiently. The inverse function enables us to compute special states from output prefixes even when they are not actually stored on the hard disk, with various combinations of time T and memory M satisfying $M\sqrt{T} = 2^{48}$. If we choose $M = 2^{36}$, we get $T = 2^{24}$, and thus we can carry out the attack in a few minutes, after a 2^{48} preprocessing stage which explores the structure of this function f.

Key idea 10: A5/1 is very efficient on a PC. The A5/1 algorithm was designed to be efficient in hardware, and its straightforward software implementation is quite slow. To execute the preprocessing stage, we have to run it on a distributed network of PC's up to 2^{48} times, and thus we need an extremely efficient way to compute the effect of one clock cycle on the three registers.

We exploit the following weakness in the design of A5/1: Each one of the three shift registers is so small that we can precompute all its possible states, and keep them in RAM as three cyclic arrays, where successive locations in each array represent successive states of the corresponding shift register. In fact, we don't have to keep the full states in the arrays, since the only information we have to know about a state is its clocking tap and its output tap. A state can thus be viewed as a triplet of indices (i, j, k) into three large single bit arrays (see Figure 2). $A_1(i), A_2(j), A_3(k)$ are the clocking taps of the current state, and $A_1(i-11), A_2(j-12), A_3(k-13)$ are the output taps of the current state (since these are the corresponding delays in the movement of clocking taps to output taps when each one of the three registers is clocked). Since there is no mixing of the values of the three registers, their only interaction is in determining which of the three indices should be incremented by 1. This can be determined by a precomputed table with three input bits (the clocking taps) and three output bits (the increments of the three registers). When we clock A5/1 in our software implementation, we don't shift registers or compute feedbacks - we just add a 0/1 vector to the current triplet of indices. A typical two dimensional variant of such movement vectors in triplet space is described in Figure 3. Note the local tree structure determined by the deterministic forward evaluation and the nondeterministic backward exploration in this triplet representation.

Since the increment table is so small, we can expand the A tables from bits to bytes, and use a larger precomputed table with 2^{24} entries, whose inputs are the three bytes to the right of the clocking taps in the three registers, and outputs are the three increments to the indices which allow us to jump directly to the state which is 8 clock cycles away. The total amount of RAM needed for the state arrays and precomputed movement tables is less than 128 MB, and the total cost of advancing the three registers for 8 clock cycles is one table lookup and three integer additions! A similar table lookup technique can be used to compute in a single step output bytes instead of output bits, and to speed up the process of running A5/1 backwards.

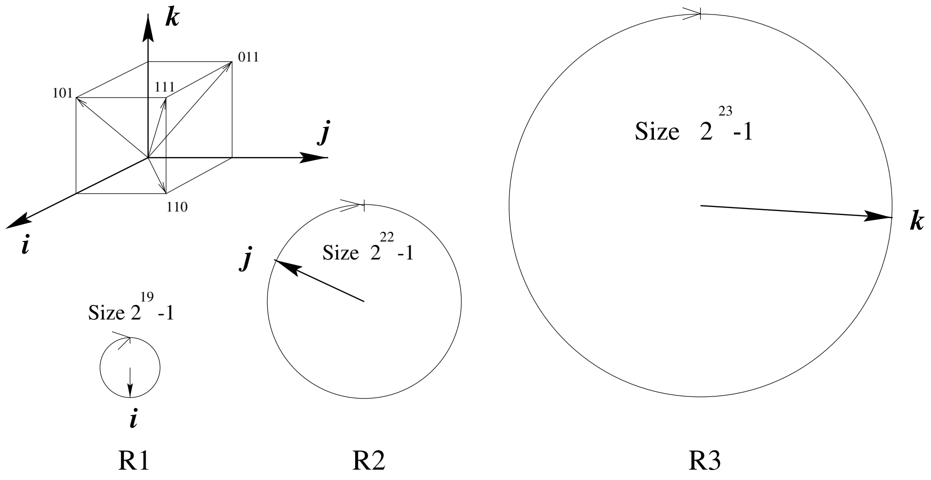


Fig. 2. Triplet representation of a state.

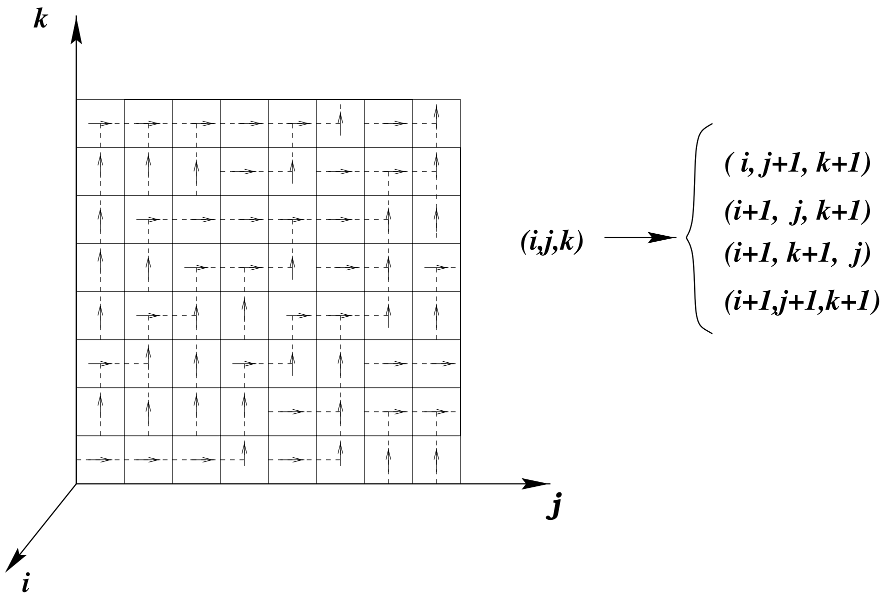


Fig. 3. The state-transition graph in the triplet representation of A5/1.

5 Detailed Description of the Attacks

In this section we fill in the missing details, and analyse the success rate of the new attacks.

5.1 Efficient Sampling of Special States

Let α be any 16 bit pattern of bits. To simplify the analysis, we prefer to use an α which does not coincide with shifted versions of itself (such as $\alpha = 1000...0$) since this makes it very unlikely that a single 228-bit frame contains more than one occurrence of α.

The total number of states which generate an output prefix of α is about $2^{64} * 2^{-16} = 2^{48}$. We would like to generate all of them in a (barely doable) 2^{48} preprocessing stage, without trying all the 2^{64} possible states and discarding the vast majority which fail the test. The low sampling resistance of A5/1 is made possible by several flaws in its design, which are exploited in the following algorithm:

– Pick an arbitrary 19-bit value for the shortest register $R1$. Pick arbitrary values for the rightmost 11 bits in $R2$ and $R3$ which will enter the clock control taps in the next few cycles. We can thus define $2^{19+11+11} = 2^{41}$ partial states.

– For each partial state we can uniquely determine the clock control of the three registers for the next few cycles, and thus determine the identity of the bits that enter their msb's and affect the output.
– Due to the majority clock control, at least one of $R2$ and $R3$ shifts a new (still unspecified) bit into its msb at each clock cycle, and thus we can make sure that the computed output bit has the desired value. Note that about half the time only one new bit is shifted (and then its choice is forced), and about half the time two new bits are shifted (and then we can choose them in two possible ways). We can keep this process alive without time consuming trial and error as long as the clock control taps contain only known bits whereas the output taps contain at least one unknown bit. A5/1 makes this very easy, by using a single clocking tap and placing it in the middle of each register: We can place in $R2$ and $R3$ 11 specified bits to the right of the clock control tap, and 11-12 unspecified bits to the right of the output tap. Since each register moves only 3/4 of the time, we can keep this process alive for about 16 clock cycles, as desired.
– This process generates only special states, and cannot miss any special state (if we start the process with its partial specification, we cannot get into an early contradiction). We can similarly generate any number $c < 2^{48}$ of randomly chosen special states in time proportional to c. As explained later in the paper, this can make the preprocessing faster, at the expense of other parameters in our attack.

5.2 Efficient Disk Probing

To leave room for a sufficiently long identifying prefix of 35 bits after the 16-bit α, we allow it to start only at bit positions 1 to 177 in each one of the given frames (i.e., at a distance of 101 to 277 from the initial state). The expected number of occurrences of α in the data produced by A5/1 during a two minute conversation is thus $2^{-16} * 177 * 120 * 1000/4.6 \approx 71$. This is the expected number of times b we access the hard disk. Since each random access takes about 6 milliseconds, the total disk access time becomes negligible (about 0.4 seconds).

5.3 Efficient Disk Storage

The data items we store on the disk are (prefix, state) pairs. The state of A5/1 contains 64 bits, but we keep only special states and thus we can encode them efficiently with shorter 48 bit names, by specifying the 41 bits of the partial state and the ≈ 7 choice bits in the sampling procedure. We can further reduce the state to less than 40 bits (5 bytes) by leaving some of the 48 bits unspecified. This saves a considerable fraction of the disk space prepared during preprocessing, and the only penalty is that we have to try a small number of candidate states instead of one candidate state for each one of the 71 relevant frames. Since this part is so fast, even in its slowed down version it takes less than a second.

The output prefix produced from each special state is nominally of length 16+35=51 bits. However, the first 16 bits are always the constant α, and the next 35 bits are stored in sorted order on the disk. We can thus store the full value of these 35 bits only once per sector, and encode on the disk only their small increments (with a default value of 1). Other possible implementations are to use the top parts of the prefixes as direct sector addresses or as file names. With these optimizations, we can store each one of the sorted (prefix, state) pairs in just 5 bytes. The largest commercially available PC hard disks (such as IBM Ultrastar 72 ZX or Seagate Cheetah 73) have 73 gigabytes. By using two such disks, we can store $146 * 2^{30}/5 \approx 2^{35}$ pairs during the preprocessing stage, and characterize each one of them by the (usually unique) 35-bit output prefix which follows α.

5.4 Efficient Tree Exploration

The forward state-transition function of A5/1 is deterministic, but in the reverse direction we have to consider four possible predecessors. About 3/8 of the states have no predecessors, 13/32 of the states have one predecessor, 3/32 of the states have two predecessors, 3/32 of the states have three predecessors, and 1/32 of the states have four predecessors.

Since the average number of predecessors is 1, Golic assumed that a good statistical model for the generated trees of predecessors is the critical branching process (see [3]). We were surprised to discover that in the case of A5/1, there was a very significant difference between the predictions of this model and our experimental data. For example, the theory predicted that only 2% of the states would have some predecessor at depth 100, whereas in a large sample of 100,000,000 trees we generated from random A5/1 states the percentage was close to 15%. Another major difference was found in the tail distributions of the number of sons at depth 100: Theory predicted that in our sample we should see some cases with close to 1000 sons, whereas in our sample we never saw trees with more than 120 sons at depth 100.

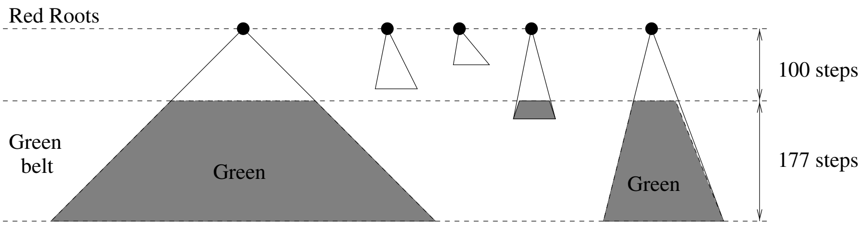


Fig. 4. Trees of different sizes.

5.5 The Biased Birthday Attack

To analyse the performance of our biased birthday attack, we introduce the following notation:

Definition 1 *A state s is coloured* **red***, if the sequence of output bits produced from state s starts with α (i.e., it is a special state). The subspace of all the red states is denoted by R.*

Definition 2 *A state is coloured* **green***, if the sequence of output bits produced from state s contains an occurrence of α which starts somewhere between bit positions 101 and 277. The subspace of all the green states is denoted by G.*

The red states are the states that we keep in the disk, look for in the data, and try to collide by comparing their prefixes. The green states are all the states that could serve as initial states in frames that contain α. Non-green initial states are of no interest to us, since we discard the frames they generate from the actual data.

The size of R is approximately 2^{48}, since there are 2^{64} possible states, and the probability that α occurs right at the beginning of the output sequence is 2^{-16}. Since the redness of a state is not directly related to its separate coordinates i, j, k in the triplet space, the red states can be viewed as randomly and sparsely located in this representation. The size of G is approximately $177 * 2^{48}$ (which is still a small fraction of the state space) since α has 177 opportunities to occur along the output sequence.

Since a short path of length 277 in the output sequence is very unlikely to contain two occurrences of α, the relationship between green and red states is essentially many to one: The set of all the relevant states we consider can be viewed as a collection of disjoint trees of various sizes, where each tree has a red state as its root and a "belt" of green states at levels 101 to 277 below it (see Figure 4). The weight $W(s)$ of a tree whose root is the red state s is defined as the number of green states in its belt, and s is called k-heavy if $W(s) \geq k$.

The crucial observation which makes our biased birthday attack efficient is that in A5/1 there is a huge variance in the weights of the various red states. We ran the tree exploration algorithm on 100,000,000 random states and computed their weights. We found out that the weight of about 85% of the states was zero, because their trees died out before reaching depth 100. Other weights ranged all the way from 1 to more than 26,000.

The leftmost graph of Figure 5 describes for each x which is a multiple of 100 the value y which is the total weight of all the trees whose weights were between x and $x + 100$. The total area under the graph to the right of $x = k$ represents the total number of green states in all the k-heavy trees in our sample.

The initial mixing of the key and frame number, which ignores the usual clock control and flips the least significant bits of the registers about half the time before shifting them, can be viewed as random jumps with uniform probability distribution into new initial states: even a pair of frame counters with Hamming distance 1 can lead to far away initial states in the triplet space. When we restrict our attention to the frames that contain α, we get a uniform probability distribution over the green states, since only green states can serve as initial states in such frames.

The red states, on the other hand, are not encountered with uniform probability distribution in the actual data. For example, a red state whose tree has no green belt will never be seen in the data. On the other hand, a red state with a huge green belt has a huge number of chances to be reached when the green initial state is chosen with uniform probability distribution. In fact the probability of encountering a particular red state s in a particular frame which is known to contain α is the ratio of its weight $W(s)$ and the total number of green states $177 * 2^{48}$, and the probability of encountering it in one of the 71 relevant frames is $P_B(s) = 71 * W(s)/(177 * 2^{48})$.

Since $P_B(s)$ has a huge variance, we can maximize the expected number of collisions $\sum_s P_A(s) * P_B(s)$ by choosing red points for the hard disk not with uniform probability distribution, but with a biased probability $P_A(s)$ which maximizes the correlation between these distributions, while minimizing the expected size of A. The best way to do this is to keep on the disk only the heaviest trees. In other words, we choose a threshold number k, and define $P_A(s) = 0$ if $W(s) < k$, and $P_A(s) = 1$ if $W(s) \geq k$. We can now easily compute the expected number of collisions by the formula:

$$\sum_s P_A(s) * P_B(s) = \sum_{s|W(s)\geq k} 71 * W(s)/(177 * 2^{48})$$

which is just the number of red states we keep on the disk, times the average weight of their trees, times $71/(177 * 2^{48})$.

In our actual attack, we keep 2^{35} red states on the disk. This is a 2^{-13} fraction of the 2^{48} red states. With such a tiny fraction, we can choose particularly heavy trees with an average weight of 12,500. The expected number of colliding red states in the disk and the actual data is $2^{35} * 12,500 * 71/(177 * 2^{48}) \approx 0.61$. This expected value makes it quite likely that a collision will actually exist. [2]

The intuition behind the biased time memory tradeoff attack is very simple. We store red states, but what we really want to collide are the green states in their belts (which are accessible from the red roots by an easy computation). The 71 green states in the actual data are uniformly distributed, and thus we want to cover about 1% of the green area under the curve in the right side of Figure 5. Standard time memory tradeoff attacks store random red states, but each stored state increases the coverage by just 177 green states on average. With our optimized choice in the preprocessing stage, each stored state increases the coverage by 12,500 green states on average, which improves the efficiency of the attack by almost two orders of magnitude.

5.6 Efficient Determination of Initial States

One possible disadvantage of storing heavy trees is that once we find a collision, we have to try a large number of candidate states in the green belt of the colliding

[2] Note that in time memory tradeoff attacks, it becomes increasingly expensive to push this probability towards 1, since the only way to guarantee success is to memorize the whole state space.

red state. Since each green state is only partially specified in our compact 5-byte representation, the total number of candidate green states can be hundreds of thousands, and the real time part of the attack can be relatively slow.

However, this simple estimate is misleading. The parasitic red states obtained from the partial specification can be quickly discarded by evaluating their outputs beyond the guaranteed occurrence of α and comparing it to the bits in the given frame. In addition, we know the exact location of α in this frame, and thus we know the exact depth of the initial state we are interested in within the green belt. As a result, we have to try only about 70 states in a cut through the green belt, and not the 12,500 states in the full belt.

5.7 Reducing the Preprocessing Time of the Biased Birthday Attack

The 2^{48} complexity of the preprocessing stage of this attack can make it too time consuming for a small network of PC's. In this section we show how to reduce this complexity by any factor of up to 1000, by slightly increasing either the space complexity or the length of the attacked conversation.

The efficient sampling procedure makes it possible to generate any number $c < 2^{48}$ of random red states in time proportional to c. To store the same number of states in the disk, we have to choose a larger fraction of the tested trees, which have a lower average weight, and thus a less efficient coverage of the green states. Table 2 describes the average weight of the heaviest trees for various fractions of the red states, which was experimentally derived from our sample of 100,000,000 A5/1 trees. This table can be used to choose the appropriate value of k in the

Table 2. The average weight of the heaviest trees for various fractions of R.

Average Weights							
2^{-4}	2432	2^{-5}	3624	2^{-6}	4719	2^{-7}	5813
2^{-8}	6910	2^{-9}	7991	2^{-10}	9181	2^{-11}	10277
2^{-12}	11369	2^{-13}	12456	2^{-14}	13471	2^{-15}	14581
2^{-16}	15686	2^{-17}	16839	2^{-18}	17925	2^{-19}	19012
2^{-20}	20152	2^{-21}	21227	2^{-22}	22209	2^{-23}	23515
2^{-24}	24597	2^{-25}	25690	2^{-26}	26234		

definition the k-heavy trees for various choices of c. The implied tradeoff is very favorable: If we increase the fraction from 2^{-13} to 2^{-7}, we can reduce the preprocessing time by a factor of 64 (from 2^{48} to 2^{42}), and compensate by either doubling the length of the attacked conversation from 2 minutes to 4 minutes, or doubling the number of hard disks from 2 to 4. The extreme point in this tradeoff is to store in the disk all the sampled red states with nonzero weights (the other sampled red states are just a waste of space, since they will never be seen in the actual data). In A5/1 about 15% of the red states have nonzero weights, and thus we have to sample about 2^{38} red states in the preprocessing stage in order to find the 15% among them (about 2^{35} states) which we want to store, with an average tree weight of 1180. To keep the same probability of success, we have to attack conversations which last about half an hour.

A further reduction in the complexity of the preprocessing stage can be obtained by the early abort strategy: Explore each red state to a shallow depth, and continue to explore only the most promising candidates which have a large number of sons at that depth. This heuristic does not guarantee the existence of a large belt, but there is a clear correlation between these events.

To check whether the efficiency of our biased birthday attack depends on the details of the stream cipher, we ran several experiments with modified variants of A5/1. In particular, we concentrated on the effect of the clock control rule, which determines the noninvertibility of the model. For example, we hashed the full state of the three registers and used the result to choose among the four possible majority-like movements (+1,+1,+1), (+1,+1,0), (+1,0,+1), (0,+1,+1) in the triplet space. The results were very different from the real majority rule. We then replaced the majority rule by a minority rule (if all the clocking taps agree, all the registers move, otherwise only the minority register moves). The results of this minority rule were very similar to the majority-like hashing case, and very different from the real majority case (see Figure 5). It turns out that in this sense A5/1 is actually stronger than its modified versions, but we do not currently understand the reason for this strikingly different behavior. We believe that the type of data in Table 2, which we call the **tail coverage** of the cryptosystem, can serve as a new security measure for stream ciphers with noninvertible state transition functions.

5.8 Extracting the Key from a Single Red State

The biased birthday attack was based on a direct collision between a state in the disk and a state in the data, and required ≈ 71 red states from a relatively long (≈ 2 minute) prefix of the conversation. In the random subgraph attack we use indirect collisions, which make it possible to find the key with reasonable probability from the very first red state we encounter in the data, even though it is unlikely to be stored in the disk. This makes it possible to attack A5/1 with less than two seconds of available data. The actual attack requires several minutes instead of one second, but this is still a real time attack on normal telephone conversations.

The attack is based on Hellman's original time-memory tradeoff for block ciphers, described in [4]. Let E be an arbitrary block cipher, and let P be some fixed plaintext. Define the function f from keys K to ciphertexts C by $f(K) = E_K(P)$. Assuming that all the plaintexts, ciphertexts and keys have the same binary size, we can consider f as a random function (which is not necessarily one-to-one) over a common space U. This function is easy to evaluate and to iterate but difficult to invert, since computing the key K from the ciphertext $f(K) = E_K(P)$ is essentially the problem of chosen message cryptanalysis.

Hellman's idea was to perform a precomputation in which we choose a large number m of random start points in U, and iterate f on each one of them t times. We store the m (start point, end point) pairs on a large disk, sorted into increasing endpoint order. If we are given $f(K)$ for some unknown K which is located somewhere along one of the covered paths, we can recover K by repeatedly applying f in the easy forward direction until we hit a stored end point, jump to its corresponding start point, and continue to apply f from there. The last point before we hit $f(K)$ again is likely to be the key K which corresponds to the given ciphertext $f(K)$.

Since it is difficult to cover a random graph with random paths in an efficient way, Hellman proposed a rerandomization technique which creates multiple variants of f (e.g., by permuting the order of the output bits of f). We use t variants f_i, and iterate each one of them t times on m random start points to get m corresponding end points. If the parameters m and t satisfy $mt^2 = |U|$, then each state is likely to be covered by one of the variants of f. Since we have to handle each variant separately (both in the preprocessing and in the actual attack), the total memory becomes $M = mt$ and the total running time becomes $T = t^2$, where M and T can be anywhere along the tradeoff curve $M\sqrt{T} = |U|$. In particular, Hellman suggests using $M = T = |U|^{2/3}$.

A straightforward application of this $M\sqrt{T} = |U|$ tradeoff to the $|U| = 2^{64}$ states of A5/1 with the maximal memory $M = 2^{36}$ requires time $T = 2^{56}$, which is much worse than previously known attacks. The basic idea of the new random subgraph attack is to apply the time-memory tradeoff to the subspace R of 2^{48} red states, which is made possible by the fact that it can be efficiently sampled. Since T occurs in the tradeoff formula $M\sqrt{T} = |U|$ with a square root, reducing the size of the graph by a modest 2^{16} (from $|U| = 2^{64}$ to $|R| = 2^{48}$) and using the same memory ($M = 2^{36}$), reduces the time by a huge factor of 2^{32} (from $T = 2^{56}$ to just $T = 2^{24}$). This number of steps can be carried out in several minutes on a fast PC.

What is left is to design a random function f over R whose output-permuted variants are easy to evaluate, and for which the inversion of any variant yields the desired key. Each state has a "full name" of 64 bits which describes the contents of its three registers. However, our efficient sampling technique enables us to give each red state a "short name" of 48 bits (which consists of the partial contents of the registers and the random choices made during the sampling process), and to quickly translate short names to full names. In addition, red states are characterized (almost uniquely) by their "output names" defined as the 48 bits which occur after α in their output sequences. We can now define the desired function f over 48-bit strings as the mapping from short names to output names of red states: Given a 48-bit short name x, we expand it to the full name of a red state, clock this state 64 times, delete the initial 16-bit α, and define $f(x)$ as the remaining 48 output bits. The computation of $f(x)$ from x can be efficiently done by using the previously described precomputed tables, but the computation of x from $f(x)$ is exactly the problem of computing the (short) name of an unknown red state from the 48 output bits it produces after

α. When we consider some output-permuted variant f_i of f, we obviously have to apply the same permutation to the given output sequence before we try to invert f_i over it.

The recommended preprocessing stage stores 2^{12} tables on the hard disk. Each table is defined by iterating one of the variants f_i 2^{12} times on 2^{24} randomly chosen 48-bit strings. Each table contains 2^{24} (start point, end point) pairs, but implicitly covers about 2^{36} intermediate states. The collection of all the 2^{12} tables requires 2^{36} disk space, but implicitly covers about 2^{48} red states.

The simplest implementation of the actual attack iterates each one of the 2^{12} variants of f separately 2^{12} times on appropriately permuted versions of the single red state we expect to find in the 2 seconds of data. After each step we have to check whether the result is recorded as an end point in the corresponding table, and thus we need $T = 2^{24}$ probes to random disk locations. At 6 ms per probe, this requires more than a day. However, we can again use Rivest's idea of special points: We say that a red state is bright if the first 28 bits of its output sequence contain the 16-bit α extended by 12 additional zero bits. During preprocessing, we pick a random red start point, and use f_i to quickly jump from one red state to another. After approximately 2^{12} jumps, we expect to encounter another bright red state, at which we stop and store the pair of (start point, end point) in the hard disk. In fact, each end point consists of a 28 bit fixed prefix followed by 36 additional bits. As explained in the previous attack, we do not have to store either the prefix (which is predictable) or the suffix (which is used as an index) on the hard disk, and thus we need only half the expected storage. We can further reduce the required storage by using the fact that the bright red states have even shorter short names than red states (36 instead of 48 bits), and thus we can save 25% of the space by using bright red instead of red start points in the table. [3] During the actual attack, we find the first red state in the data, iterate each one of the 2^{12} variants of f over it until we encounter a bright red state, and only then search this state among the pairs stored in the disk. We thus have to probe the disk only once in each one of the $t = 2^{12}$ tables, and the total probing time is reduced to 24 seconds.

There are many additional improvement ideas and implementation details which will be described in the final version of this paper.

Acknowledgements. We would like to thank Ross Anderson, Mike Roe, Jovan Golic, Marc Briceno, and Ian Goldberg for their pioneering contributions to the analysis of A5/1, which made this paper possible. We would like to thank Marc Briceno, Ian Goldberg, Ron Rivest and Nicko van Someren for sending very useful comments on the early version of this paper.

[3] Note that we do not know how to jump in a direct way from one bright red state to another, since we do not know how to sample them in an efficient way. We have to try about 2^{12} red states in order to find one bright red start point, but the total time needed to find the 2^{36} bright red start points in all the tables is less than the 2^{48} complexity of the path evaluations during the preprocessing stage.

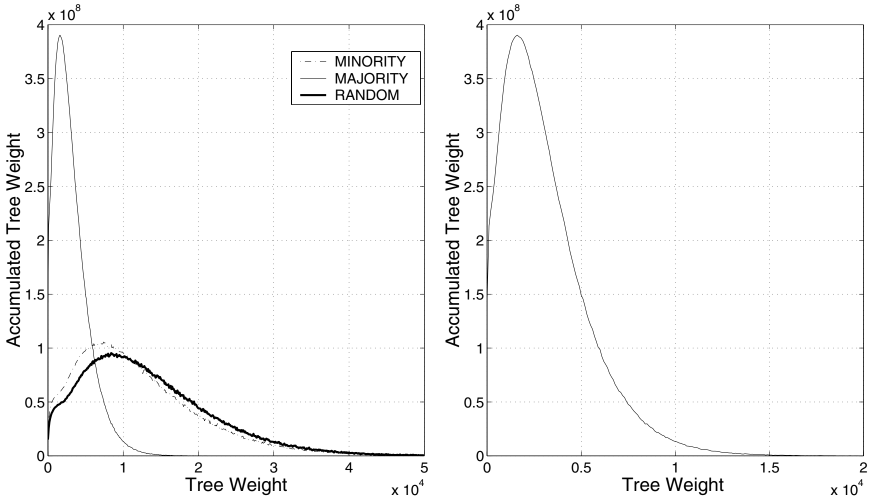


Fig. 5. Weight distributions. The graph on the left shows weight distribution for the majority function; the graph on the right compares the weight distributions of several clock-control functions.

References

1. R. Anderson, M. Roe, *A5*, `http://jya.com/crack-a5.htm`, 1994.
2. S. Babbage, *A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers*, European Convention on Security and Detection, IEE Conference publication, No. 408, May 1995.
3. M. Briceno, I. Goldberg, D. Wagner, A pedagogical implementation of A5/1, `http://www.scard.org`, May 1999.
4. J. Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, proceedings of EUROCRYPT'97, LNCS 1233,pp.239–255, Springer-Verlag 1997.
5. M. E. Hellman, *A Cryptanalytic Time-Memory Trade-Off*, IEEE Transactions on Information Theory, Vol. IT-26, N 4, pp.401–406, July 1980.

Statistical Analysis of the Alleged RC4 Keystream Generator

Scott R. Fluhrer and David A. McGrew

Cisco Systems, Inc.
170 West Tasman Drive, San Jose, CA 95134
{sfluhrer, mcgrew}@cisco.com

Abstract. The alleged RC4 keystream generator is examined, and a method of explicitly computing digraph probabilities is given. Using this method, we demonstrate a method for distinguishing 8-bit RC4 from randomness. Our method requires less keystream output than currently published attacks, requiring only $2^{30.6}$ bytes of output. In addition, we observe that an attacker can, on occasion, determine portions of the internal state with nontrivial probability. However, we are currently unable to extend this observation to a full attack.

1 Introduction

We show an algorithm for deriving the exact probability of a digraph in the output of the alleged RC4 stream cipher. This algorithm has a running time of approximately 2^{6n}, where n is the number of bits in a single output. Using the computed probabilities of each digraph for the case that $n = 5$, we discern which digraphs have probabilities furthest from the value expected from a uniform random distribution of digraphs. Extrapolating this knowledge, we show how to distinguish the output of the alleged RC4 cipher with $n = 8$ from randomness with $2^{30.6}$ outputs. This result improves on the best known method of distinguishing that cipher from a truly random source. In addition, heuristic arguments about the cause of the observed anomalies in the digraph distribution are offered.

The irregularities in the digraph distribution that we observed allow the recovery of n and i parameters (defined in Section 2) if the attacker happens not to know them. Also, an attacker can use this information in a ciphertext-only attack, to reduce the uncertainty in a highly redundant unknown plaintext.

We also observe how an attacker can learn, with nontrivial probability, the value of some internal variables at certain points by observing large portions of the keystream. We are unable to derive the entire state from this observation, though with more study, this insight might lead to an exploitable weakness in the cipher.

This paper is structured as follows. In Section 2, the alleged RC4 cipher is described, and previous analysis and results are summarized. Section 3 presents our analysis of that cipher, and Section 4 investigates the mechanisms behind the statistical anomalies that we observe in that cipher. Section 5 examines fortuitous states, which allow the attacker to deduce parts of the internal state. In Section 6, extensions of our analysis and directions for future work are discussed. Section 7 summarizes our conclusions. Lastly, the Appendix summarizes the results from information theory that are needed to put a strong bound the effectiveness of tests based on the statistical anomalies, and presents those bounds for our work and for previous work.

2 Description of the Alleged RC4 Cipher and Other Work

The alleged RC4 keystream generator is an algorithm for generating an arbitrarily long pseudorandom sequence based on a variable length key. The pseudorandom sequence is conjectured to be cryptographically secure for use in a stream cipher. The algorithm is parameterized by the number of bits n within a permutation element, which is also the number of bits that are output by a single iteration of the next state function of the cipher. The value of $n = 8$ is of greatest interest, as this is the value used by all known RC4 applications.

The RC4 keystream generator was created by RSA Data Security, Inc. [6]. An anonymous source claimed to have reverse-engineered this algorithm, and published an alleged specification of it in 1994 [8]. Although public confirmation of the validity of this specification is still lacking, we abbreviate the name 'alleged RC4' to 'RC4' in the remainder of this paper. We also denote n-bit RC4 as RC4/n.

A summary of the RC4 operations is given in Table 1. Note that in this table, and throughout this paper, all additions and increments are done modulo 2^n.

Table 1. The RC4 next state function. i and j are elements of $\mathbb{Z}/2^n$, and S is a permutation of integers between zero and $2^n - 1$. All increments and sums are modulo 2^n.

1. Increment i by 1
2. Increment j by $S[i]$
3. Swap $S[i]$ and $S[j]$
4. Output $S[S[i] + S[j]]$

2.1 Previous Analysis of RC4

The best previously known result for distinguishing the output of RC4 from that of a truly random source was found by Golić [3,2], who presents a statistical defect that he estimates will allow an attacker to distinguish RC4/8 from randomness with approximately 2^{40} successive outputs. However, this result appears to be somewhat optimistic. We use the information theoretic lower bound on the number of bytes needed to distinguish RC4 from randomness, for a given statistical anomaly, and use this to measure the effectiveness of Golić's anomaly and our own anomalies (see the Appendix). The number of bytes of RC4/8 output needed to reduce the false positive and false negative rates to 10% is $2^{44.7}$, using Golić's anomaly, while the irregularities in the digraph distribution that we found require just $2^{30.6}$ bytes to achieve the same result.

Mister and Tavares analyzed the cycle structure of RC4 [5]. They observe that the state of the permutation can be recovered, given a significant fraction of the full keystream. In addition, they also present a backtracking algorithm that can recover the permutation from a short keystream output. Their analyses are supported by experimental results on RC4/n for $n < 6$, and show that an RC4/5 secret key can be recovered after only 2^{42} steps, though the nominal key size is about 160 bits.

Knudsen et. al. presented attacks on weakened versions of RC4 [4]. The weakened RC4 variants that they studied change their internal state less often than does RC4, though they change it in a similar way. Their basic attack backtracks through the internal state, guessing values of table entries that have not yet been observed, and backtracking upon contradictions. They present several variants of their attack, and analyze its runtime. They estimate that the complexity of their attack is less than the square root of the number of possible RC4 states.

3 Analysis of Digraph Probabilities

The probability with which each digraph (that is, each successive pair of n-bit outputs) will appear in the output of RC4 is directly computable, given some reasonable assumptions. The probability of each digraph for each value of the i index is also computable. By taking advantage of the information on i, rather than averaging over all values of i and allowing some of the detail about the statistical anomalies to wash away, it is possible to more effectively distinguish RC4 from randomness.

To simplify analysis, we idealize the key set up phase. We assume that the key setup will generate each possible permutation with equal probability, and will assign all possible values to j with equal probability. Then, after any fixed N steps, all states of j and the permutation will still have equally probability, because the next state function is invertible. This is an idealization; the actual RC4 key setup will initialize j to zero. Also, the RC4 key setup routine generates only 2^{n_k} different permutations, where n_k is the number of bits in the key, while there are $2^n!$ possible permutations. Intuitively, our idealization becomes a close approximation of the internal state after RC4 runs for a short period of time.

However, we leave in the assumption that the i pointer is initially zero after the key setup phase. Note that, since each step changes i in a predictable manner, the attacker can assume knowledge of the i pointer for each output.

We compute the exact digraph probabilities, under the assumptions given above, by counting the number of internal states consistent with each digraph. This approach works with RC4 because only a limited amount of the unknown internal state actually affects each output, though the total amount of internal state is quite large.

Starting at step 4 of Table 1, we look at what controls the two successive outputs. The exhaustive list of everything on which those two outputs depend on is given in Table 2.

Table 2. The variables that control two successive outputs of RC4 and the cryptanalyst's knowledge of them.

Variable	**Cryptanalyst's knowledge**
i	known (increments regularly)
j	unknown
$S[i]$	unknown
$S[j]$	unknown
$S[S[i]+S[j]]$	known (first output)
$S[i+1]$	unknown
$S[j+S[i+1]]$	unknown
$S[j+S[i+1]]$ if $i+1=S[i+1]+S[j+S[i+1]]$ $S[i+1]$ if $j+S[i+1]=S[i+1]+S[j+S[i+1]]$ $S[S[i+1]+S[j+S[i+1]]]$ otherwise	known (second output)

As the next-state algorithm progresses, for each successive unknown value, any value that is consistent with the previously seen states is equally probable. Thus the probability of a digraph (a, b) for a particular value of i can be found by stepping through all possible values of all other variables, and counting the number of times that each output is consistent with the fixed values of i, a, and b. The consistency of a set of values is determined by the fact that S is a permutation. Because the start states were considered equally probable, this immediately gives us the exact value of the probability of $i, (a, b)$. This approach requires about 2^{5n} operations to compute the probability of a single digraph, for a given value of i, as there are five n-bit unknowns in Table 2. Approximately 2^{8n} operations are required to compute the probabilities of a digraph for all values of i. This puts the most interesting case of $n = 8$ out of immediate reach, with a computational cost of 2^{64} operations. However, we circumvented this difficulty by computing the exact $n = 3, 4, 5$ digraph distributions for all i, observing which digraphs have anomalous probabilities, and estimating the probabilities of the anomalous digraphs for RC4/8. This method is described in the next two subsections.

3.1 Anomalous RC4 Outputs

The full digraph distributions for $n = 3, 4$, and 5 are computable with about 2^{40} operations. We computed these, and found that the distributions were significantly different from a uniform distribution. In addition, there is a consistency (across different values of n) to the irregularities in the digraph probabilities. In particular, one type of digraph is more probable than expected by a factor of approximately $1 + 2^{-n+1}$, seven types of digraphs are more probable than expected by approximately $1 + 2^{-n}$, and three types of digraphs are less probable than expected by approximately $1+2^{-n}$. These results are summarized in Table 3.

Table 3. Positive and negative events. Here, i is the value of the index when the first symbol of the digraph the output. The top eight digraphs are in the set of positive events, and the bottom three digraphs are in the set of negative events, as defined in Section 3.1. The probabilities are approximate.

Digraph	**Value(s) of i**	**Probability**
$(0, 0)$	$i = 1$	$2^{-2n}(1 + 2^{-n+1})$
$(0, 0)$	$i \neq 1, 2^n - 1$	$2^{-2n}(1 + 2^{-n})$
$(0, 1)$	$i \neq 0, 1$	$2^{-2n}(1 + 2^{-n})$
$(i + 1, 2^n - 1)$	$i \neq 2^n - 2$	$2^{-2n}(1 + 2^{-n})$
$(2^n - 1, i + 1)$	$i \neq 1, 2^n - 2$	$2^{-2n}(1 + 2^{-n})$
$(2^n - 1, i + 2)$	$i \neq 0, 2^n - 1, 2^n - 2, 2^n - 3$	$2^{-2n}(1 + 2^{-n})$
$(2^n - 1, 0)$	$i = 2^n - 2$	$2^{-2n}(1 + 2^{-n})$
$(2^n - 1, 1)$	$i = 2^n - 1$	$2^{-2n}(1 + 2^{-n})$
$(2^n - 1, 2)$	$i = 0, 1$	$2^{-2n}(1 + 2^{-n})$
$(2^{n-1} + 1, 2^{n-1} + 1)$	$i = 2$	$2^{-2n}(1 + 2^{-n})$
$(2^n - 1, 2^n - 1)$	$i \neq 2^n - 2$	$2^{-2n}(1 - 2^{-n})$
$(0, i + 1)$	$i \neq 0, 2^n - 1$	$2^{-2n}(1 - 2^{-n})$

We call the event that a digraph appears in the RC4 output at a given value of i a *positive event* when it is significantly more probable than expected. A *negative event* is similarly defined to be the appearance of a digraph at a given i that is significantly less probable than expected. An exhaustive list of positive and negative events is provided in Table 3.

In Section 4, we examine these particular digraphs to see why they are more or less likely than expected. Most of the positive events correspond to length 2 fortuitous states, which will be defined in Section 5. For the $(0, 1)$ and $(0, 0)$ positive events, and the negative events, a more complicated mechanism occurs, which is discussed in the next section.

3.2 Extrapolating to Higher Values of n

To apply our attack to higher values of n without directly computing the digraph probabilities, we computed the probabilities of positive events and negative events by running RC4/8 with several randomly selected keys and counting the occurances of those events in the RC4 output. The observed probabilities (derived using RC4/8 with 10 starting keys for a length of 2^{38} for each key), along with the computed expected probability from a truly random sequence, are given in Table 4. It is possible to distinguish between these two probability distributions with a 10% false positive and false negative rate by observing $2^{30.6}$ successive outputs using the data in Table 3 (see the Appendix).

Table 4. Comparison of event probabilities between RC4/8 and a random keystream. The listed probabilities are the probability that two successive outputs are the specified event

	Positive Events	**Negative Events**
RC4/8	0.00007630	0.00003022
Random	0.00007600	0.00003034

In order to evaluate the effectiveness of our 'extrapolation' approach, we compute the amount of keystream needed using a test based on our observed positive and negative events, and compare that to the best possible test using the exact probabilities. For RC4/5 our selected positive/negative events require $2^{18.76}$ keystream outputs, while the optimal test using the exact probabilities of all digraphs requires $2^{18.62}$ keystream outputs. These numbers agree to within a small factor, suggesting that the extrapolation approach is close to optimal.

4 Understanding the Statistical Anomalies

In this section we analyze the next state function of RC4 and show mechanisms that cause the increased (or decreased) likelihood of some of the anomalous digraphs. The figures below show the internal state of RC4 immediately before state 4 in Table 1 of the first output of the digraph. The bottom line shows the state of the permutation. Those permutation elements with a specific value are labeled with that value. Elements that are of unspecified value are labeled with the 'wildcard' symbol $*$. Ellipsis indicate unspecified numbers of unspecified elements, and elements separated by ellipsis may actually be in opposite order within the permutation. The elements pointed to by i and j are indicated by the i and j symbols appearing above them.

The mechanism that leads to the $(0, 1)$ digraph starts in the state

$*, \ldots, *, \overset{i}{1}, 0, *, \ldots, *, \overset{j}{AA}, *, \ldots$ where $AA = i$.

Following through the steps in the next-state function, the first output will be 0, and at the following step 4, be in the state

$*, \ldots, *, 1, \overset{\mathrm{i}}{AA}, *, \ldots, *, \overset{j}{0}, *, \ldots$

and output an 1. This mechanism occurs approximately 2^{-3n} of the time, and since other mechanisms output a $(0, 1)$ 2^{-2n} of the time, this accounts for the observed increase over expected.

For the $(0, 0)$ positive events, the additional mechanism starts with the following state:

$*, \ldots, *, \overset{i}{AA}, 0, *, \ldots, *, \overset{j}{BB}, *, \ldots$ where $AA = i + 1 - j$ and $BB = j$.

The negative events, on the other hand, correspond to mechanisms that normally contribute to the expected output which do not work in those particular cases. For example, a normal method of producing a repeated digraph (AA, AA) is[1]:

```
        i  j
*, ..., *, BB, -1, *, ..., *, AA, *, ...
```

Here the value AA occurs at location $BB - 1$. This outputs an AA, and steps into the state:

```
        j  i
*, ..., *, -1, BB, *, ..., *, AA, *, ...
```

This will output another AA, unless AA happens to be either BB or -1. In either case, this will output a BB. Since normal (AA, AA) pairs rely on this to achieve a near-expected rate, the lack of this mechanism for $(-1, -1)$ prevents the output once every approximately 2^{-3n} outputs, which accounts for the reduction of approximately a factor of 2^{-n} that we observe.

These mechanisms do not depend on the value of n, and so can be expected to operate in the $n = 8$ case. This supports our extrapolation approach, which assumes that the positive and negative events to still apply in that case.

5 Analysis of Fortuitous States

There are RC4 states in which only N elements of the permutation S are involved in the next N successive outputs. We call these states *fortuitous states*[2]. Since the variable i sweeps through the array regularly, it will always index N different array elements on N successive outputs (for $N \leq 2^n$). So, the necessary and sufficient condition for a fortuitous state is that the elements pointed to by j and pointed to by $S[i]+S[j]$ must come from the set of N array elements indexed by i.

An example of an $N = 3$ fortuitous state follows:

```
i       j
*, 255, 2, 1, *, *, ...,
```

1. advance i to 1
2. advance j to 2
3. swap $S[1]$ and $S[2]$
4. output $S[1] = 2$

```
   i  j
*, 2, 255, 1, *, *, ...,
```

1. advance i to 2
2. advance j to 1
3. swap $S[2]$ and $S[1]$
4. output $S[1] = 255$

[1] The symbol -1 is used as shorthand for $2^n - 1$ here and throughout the paper.
[2] Observing such a state is fortuitous for a cryptanalyst.

$$\begin{array}{llllll} & j & i & & & \\ *, & 255, & 2, & 1, & *, & *, \ldots, \end{array}$$

1. advance i to 3
2. advance j to 2
3. swap $S[3]$ and $S[2]$
4. output $S[3] = 2$

$$\begin{array}{llllll} & & j & i & & \\ *, & 255, & 1, & 2, & *, & *, \ldots, \end{array}$$

If $i = 0$ at the first step, and assuming that all permutations and settings for j are equally probable, then the above initial conditions will hold with probability $1/(256 \cdot 256 \cdot 255 \cdot 254)$. When the initial conditions hold, the output sequence will always be $(2, 255, 2)$. If RC4 outputs all trigraphs with equal probability (the results in our previous section imply that it doesn't, but we will use that as an approximation), the sequence $(2, 255, 2)$ will occur at $i = 0$ with probability $1/(256 \cdot 256 \cdot 256)$. This implies that, when the output is the sequence $(2, 255, 2)$ when $i = 0$, then this scenario caused that output approximately 1/253 of the time. In other words, if the attacker sees, at offset 0, the sequence $(2, 255, 2)$, he can guess that j was initially 3, and $S[1]$, $S[2]$, $S[3]$ was initially 255, 2 and 1, and be right a nontrivial portion of the time.

The number of fortuitous states can be found using a state-counting algorithm similar to that given above. The numbers of such states, for small N, are given in Table 5. The table lists, for each N, the number of fortuitous states that exist of that length, the logarithm (base 2) of the expected time between occurrances of any fortuitous state of that length, and the expected number within that length of false hits. By false hit, we mean those output patterns that have identical patterns as a fortuitous state, but are not caused by a fortuitous state. For example, an attacker can expect to see, in a keystream of length $2^{35.2}$, one fortuitous state of length 4 and 250 output patterns that look like fortuitous states.

Table 5. The number of fortuitous states for RC4/8, their expected occurrance rates, and their expected false hit rates.

Length	Number	Lg(Expected)	Expected False Hits
2	516	22.9	255
3	290	31.8	253
4	6540	35.2	250
5	25,419	41.3	246
6	101,819	47.2	241

It is not immediately clear how an attacker can use this information. What saves RC4 from an immediate break is that the state space is so huge that an attacker who directly guess 56 bits (which is approximately what you get with a length 6 fortuitous pattern) still has so many bits unguessed that there is no obvious way to proceed. However, it does appear to be a weakness that the attacker can guess significant portions of internal state at times with nontrivial probability.

It may be possible to improve the backtracking approaches to deriving RC4 state [5,4] using fortuitous states. For example, a backtracking algorithm can be started at the keystream location immediately after a fortuitous state, using the values of the internal state that are suggested by that state. This approach extends slightly the attack using ‘special streams’ presented in Section 4.3 of [4].

6 Directions for Future Work

Some extensions of our current work are possible. One direct extension is to compute the exact digraph probabilities for the case that $n = 8$, and other cases for $n > 5$. Since RC4/n is actually a complex combinatorial object, it may happen that the results for these cases are significantly different than what might be expected.

Another worthwhile direction is to investigate the statistics of trigraphs (that is, the three consecutive output symbols). The exact trigraph probabilities can be computed using an algorithm similar to that outlined in Section 3. The computational cost to compute the complete trigraph distribution, for all i, is 2^{11n}. We have computed this for RC4/4, and found that the length of outputs required to distinguish that cipher from randomness using trigraphs is about one-seventh that required when using digraphs. This result is encouraging, though it does not guarantee that trigraph statistics will be equally as effective for larger values of n. It must be considered that with $n = 4$, there are only $2^4 = 16$ entries in the table S, and that three consecutive output symbols typically uses half of the state in this cipher.

The computational cost of computing the complete trigraph distribution motivates the consideration of lagged digraphs, that is, two symbols of fixed value separated by some symbols of non-fixed value. We call the number of intervening symbols of non-fixed value the *lag*. For example, adapting the notation used above, $(1, *, 2)$ is a lag one digraph with initial value 1 and final value 2. Here we use the ‘wildcard’ symbol $*$ to indicate that the middle symbol can take on any possible value. Lagged digraphs are far easier to compute than trigraphs, because it is not necessary to individually count the states that are used only to determine the middle symbols. In general, the computational effort to compute the distribution of RC4/n lag L digraphs, for all i, requires about $2^{(8+L)n}$ operations.

Another approach to computing a digraph probability is to list the possible situations that can occur within the RC4 cipher when producing that digraph, generate the equations that must hold among the internal elements, and use algebraic means to enumerate the solutions to those equations. The number of solutions corresponds to the number of states that lead to that digraph. This approach could lead to a method to compute the exact digraph probability in a time independent of n.

Another direction would be to eliminate some of the assumptions made in our analysis. For example, the assumption that S and j are uniformly random is false, and it is especially wrong immediately after key setup. In particular, j is initially set to zero during the key setup. We venture that an analysis of fortuitous states that takes the key setup into consideration may lead to a method for deriving some information about the secret key.

7 Conclusions

We presented a method for computing exact digraph probabilities for RC4/n under reasonable assumptions, used this method to compute the exact distributions for small n, observed consistency in the digraph statistics across all values of n, and presented a simple method to extrapolate our knowledge to higher values of n. The minimum amount of RC4 output needed to distinguish that cipher from randomness was derived using information theoretic bounds, and this method was used to compare the effectiveness of our attack to those in the literature. Our methods provide the best known way to distinguish RC4/8, requiring only $2^{30.6}$ bytes of output.

While we cannot extend either attack to find the original key or the entire internal state of the cipher, further research may be able to extend these observations into an attack that is more efficient than exhaustive search.

Appendix: Information Theoretic Bounds on Distinguishing RC4 from Randomness

Information theory provides a lower bound on the number of outputs that are needed to distinguish RC4 output from a truly random sequence. We derive this bound for the case that with false positive and false negative rates of 10%, for our results and for those in the literature.

Following [1], we define the discrimination $L(p, q)$ between two probability distributions p and q as $L(p, q) = \sum_s p(s) \lg \frac{p(s)}{q(s)}$, where the sum is over all of the states in the distributions. The discrimination is the expected value of the log-likelihood ratio (with respect to the distribution p), and can be used to provide bounds on the effectiveness of hypothesis testing. A useful fact about discrimination is that in the case that l independent observations are made from the same set of states, the total discrimination is equal to l times the discrimination of a single observation.

We consider a test $\mathcal{T}$ that predicts (with some likelihood of success) whether or not a particular input string of l symbols, each of which is in $\mathbb{Z}/2^n$, was generated by n-bit RC4 or by a truly random process. If the input string was generated by RC4, the test $\mathcal{T}$ returns a 'yes' with probability $1 - \beta$. If the input string was generated by a truly random process, then $\mathcal{T}$ returns 'no' with probability $1 - \alpha$. In other words, α is the false positive rate, and β is the false negative rate. These rates can be related to the discrimination between the probability distribution p_r generated by a truly random process and the distribution p_{RC4} generated by n-bit RC4 (with a randomly selected key), where the distributions are over all possible input strings. From [1], the discrimination is related to α and β by the inequality

$$L(p_r, p_{RC4}) \geq \beta \lg \frac{\beta}{1-\alpha} + (1-\beta) \lg \frac{1-\beta}{\alpha}. \qquad (1)$$

Equality can be met by using an information-theoretic optimal test, such as a Neyman-Pearson test [1]. We expect our cryptanalyst to use such a test, and we regard Equation 1 as an equality, though the implementation of such tests are outside the scope of this paper.

Applying this result to use the RC4 digraph distribution ρ from the uniform random distribution ϕ,

$$L(\phi, \rho) = l \sum_{d \in \mathcal{D}} 2^{-2n} \lg \frac{1}{2^{2n}\rho(d)} = \beta \lg \frac{\beta}{1-\alpha} + (1-\beta) \lg \frac{1-\beta}{\alpha}, \qquad (2)$$

where $\mathcal{D}$ is the set of digraphs, and $\rho(d)$ is the probability of digraph d with respect to the distribution ρ. Solving this equation for l, we get the number of RC4 outputs needed to distinguish that cipher.

To distinguish RC4 from randomness in the case that we only know the probabilities of the positive and negative events defined in Section 3.1, we consider only the states N, P and Q, where N is the occurrance of negative event, P is the occurrance of a positive event, and Q is the occurrance of any digraph that is neither a positive nor negative event. Then the discrimination is given by Equation 1, where the sum is over these three states. Solving this equation for the number l of outputs ,with $\alpha = \beta = 0.1$ and the data from Table 4 gives $2^{30.6}$.

The linear model of RC4 derived by Golić demonstrates a bias in RC4/8 with correlation coefficient 3.05×10^{-7} [3,2]. In other words, an event that occurs after each symbol output with probability $0.5 + 1.52 \cdot 10^{-7}$ in a keystream generated by RC4, and with probability 0.5 in a keystream generated by a truly random source. Using Equation 1 with $\alpha = \beta = 0.1$, we find that at least $2^{44.7}$ bytes are required.

References

1. Blahut, R., "Principles and Practice of Information Theory", Addison-Wesley, 1983.
2. Golić, J., "Linear Models for a Time-Variant Permutation Generator", IEEE Transactions on Information Theory, vol. 45, No. 7, pp. 2374-2382, Nov. 1999
3. Golić, J., "Linear Statistical Weakness of Alleged RC4 Keystream Generator", Proceedings of EUROCRYPT '97, Springer-Verlag.
4. Knudsen, L., Meier, W., Preneel, B., Rijmen, V., and Verdoolaege, S., "Analysis Methods for (Alleged) RC4", Proceedings of ASIACRYPT '99, Springer-Verlag.

5. Mister, S. and Tavares, S., “Cryptanalysis of RC4-like Ciphers”, in the Workshop Record of the Workshop on Selected Areas in Cryptography (SAC ’98), Aug. 17-18, 1998, pp. 136-148.
6. Rivest, R., ”The RC4 encryption algorithm”, RSA Data Security, Inc, Mar. 1992
7. RSA Laboratories FAQ, Question 3.6.3, http://www.rsasecurity.com/rsalabs/faq/3-6-3.html.
8. Schneier, B., “Applied Cryptography”, New York: Wiley, 1996.

The Software-Oriented Stream Cipher SSC2

Muxiang Zhang[1], Christopher Carroll[1], and Agnes Chan[2]

[1] GTE Laboratories Inc., 40 Sylvan Road LA0MS59, Waltham, MA 02451
{mzhang, ccarroll}@gte.com
[2] College of Computer Science, Northeastern University, Boston, MA 02115
ahchan@ccs.neu.edu

Abstract. SSC2 is a fast software stream cipher designed for wireless handsets with limited computational capabilities. It supports various private key sizes from 4 bytes to 16 bytes. All operations in SSC2 are word-oriented, no complex operations such as multiplication, division, and exponentiation are involved. SSC2 has a very compact structure that makes it easy to implement on 8-,16-, and 32-bit processors. Theoretical analysis demonstrates that the keystream sequences generated by SSC2 have long period, large linear complexity, and good statistical distribution.

1 Introduction

For several reasons, encryption algorithms have been constrained in cellular and personal communications. First, the lack of computing power in mobile stations limits the use of computationally intensive encryption algorithms such as public key cryptography. Second, due to the high bit error rate of wireless channels, encryption algorithms which produce error propagation deteriorate the quality of data transmission, and hence are not well suited to applications where high bit error rates are common place. Third, the shortage of bandwidth at uplink channels (from mobile station to base station) makes encryption algorithms at low encryption (or decryption) rates unacceptable, and random delays in encryption or decryption algorithms are not desirable either. To handle these issues, the European Group Special Mobile (GSM) adopted a hardware implemented stream cipher known as alleged A5 [13]. This stream cipher has two main variants: the stronger A5/1 version and the weaker A5/2 version. Recent analysis by Biryukov and Shamir [16] has shown that the A5/1 version can be broken in less than one second on a single PC. Other than this weakness, the hardware implementation of the alleged A5 also incurs additional cost. In addition, the cost of modifying the encryption algorithm in every handset would be exorbitant when such a need is called for. For this reason, a software implemented stream cipher which is fast and secure would be preferable.

To this end, we designed SSC2, a software-oriented stream cipher which is easy to implement on 8-, 16-, and 32-bit processors. SSC2 belongs to the stream cipher family of combination generators. It combines a filtered linear feedback shift register (LFSR) and a lagged-Fibonacci generator. All operations involved in SSC2 are word-oriented, where a word consists of 4 bytes. The word sequence generated by SSC2 is added modulo-2 to the words of data frames in the manner of a Vernam cipher. SSC2 supports various private key sizes from 4 bytes to 16 bytes. It has a key scheduling scheme that stretches a private key to a master key of 21 words. The master key is loaded as the initial states of the LFSR and the lagged-Fibonacci generator. To cope with the synchronization problem, SSC2 also supplies an efficient frame key generation scheme that generates an individual key for each data frame. Theoretical analysis indicates that the keystream sequences generated by SSC2 have long period, large linear complexity, and good statistical distribution.

2 Specification of SSC2

The keystream generator of SSC2, as depicted in Figure 1, consists of a filter generator and a lagged-Fibonacci generator. In the filter generator, the LFSR is a word-oriented linear feedback shift register. The word-oriented LFSR has 4 stages with each stage containing a word. It generates a new word and shifts out an old word at every clock. The nonlinear filter compresses the 4-word content of the LFSR to a word. The lagged-Fibonacci generator has 17 stages and is also word-oriented. The word shifted out by the lagged-Fibonacci generator is left-rotated 16 bits and then added to another word selected from the 17 stages. The sum is XOR-ed with the word produced by the filter generator.

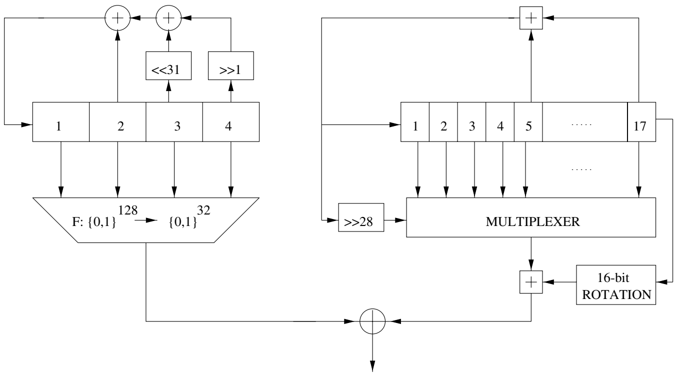


Fig. 1. The keystream generator of SSC2

2.1 The Word-Oriented Linear Feedback Shift Register

For software implementation, there are two major problems for LFSR-based keystream generators. First, the speed of a software implemented LFSR is much slower than that of a hardware implemented one. To update the state of a LFSR, a byte-oriented or word-oriented processor needs to spend many clock cycles to perform the bit-shifting and bit-extraction operations. Second, LFSR-based keystream generators usually produce one bit at every clock, which again makes the software implementation inefficient. To make the software implementation efficient, we designed a word-oriented LFSR in SSC2 by exploiting the fact that each word of a linear feedback shift register sequence can be represented as a linear transformation of the previous words of the sequence.

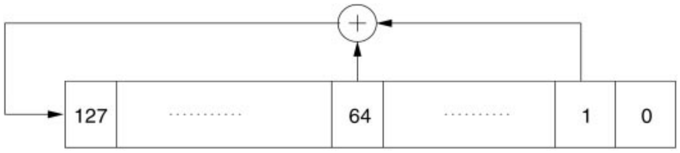


Fig. 2. The LFSR with characteristic polynomial $p(x) = x(x^{127} + x^{63} + 1)$

The LFSR used in SSC2, as depicted in Figure 4.2, has the characteristic polynomial

$$p(x) = x(x^{127} + x^{63} + 1),$$

where the factor $x^{127} + x^{63} + 1$ of $p(x)$ is a primitive polynomial over $GF(2)$. After discarding s_0, the LFSR sequence, $s_1, s_2, \ldots$, is periodic and has the least period $2^{127} - 1$. The state $S_n = (s_{n+127}, s_{n+126}, \ldots, s_n)$ at time n can be divided into 4 blocks with each block being a word, that is,

$$S_n = (x_{n+3}, x_{n+2}, x_{n+1}, x_n).$$

After running the LFSR 32 times, the LFSR has the state

$$S_{n+32} = (x_{n+4}, x_{n+3}, x_{n+2}, x_{n+1}).$$

It can be shown that

$$x_{n+4} = x_{n+2} \oplus (s_{n+32}, 0, 0, \ldots, 0) \oplus (0, s_{n+31}, s_{n+30}, \ldots, s_{n+1}). \tag{1}$$

Let $\ll$ denote the zero-fill left-shift operation. By $x \ll_j$, it means that the word x is shifted left j bits and a zero is filled to the right-most bit every time when x is shifted left 1 bit. Similarly, let $\gg$ denote the zero-fill right-shift operation. With these notations, we can rewrite equation (1) as follows

$$x_{n+4} = x_{n+2} \oplus x_{n+1} \ll_{31} \oplus x_n \gg_1, \tag{2}$$

which describes the operation of the word-oriented LFSR in Figure 1. It is interesting to note that the feedback connections of the word-oriented LFSR are not sparse even though the bit-oriented LFSR described by $p(x)$ has very sparse feedback connections.

In the bit-oriented LFSR described by Figure 2, the stage 0 is not involved in the computation of the feedback and hence is redundant. It is left there just to make the length of the LFSR to be a multiple of 32. For this reason, the content of stage 0 will be excluded from the state of the LFSR. Let S'_n denote the bit-oriented LFSR state consisting of the contents of stage 1 through stage 127, namely,

$$S'_n = (s_{n+127}, s_{n+126}, \ldots, s_{n+1}).$$

Correspondingly, let S''_n denote the state of the word-oriented LFSR, where S''_n is made up of the contents of stage 1 through stage 127 of the word-oriented LFSR at time n. Thus,

$$S''_n = S'_{32n}, \quad n \geq 0. \tag{3}$$

Proposition 1. Assume that the initial state S''_0 of the word-oriented LFSR described by (2) is not zero. Then the state sequence $S''_0, S''_1, \ldots$ is periodic and has the least period $2^{127}-1$. Furthermore, for any $0 \leq i < j < 2^{127}-1, S''_i \neq S''_j$.

Proof. Since $x^{127}+x^{63}+1$ is a primitive polynomial oner $GF(2)$, and $S'_0 = S''_0 \neq 0$, the state sequence $S'_0, S'_1, \ldots$ of the bit-oriented LFSR is periodic and has the least period $2^{127}-1$. Thus, by (3), $S''_{n+2^{127}-1} = S'_{32(n+2^{127}-1)} = S''_n$. Hence, the sequence $S''_0, S''_1, \ldots$ is periodic and has a period of $2^{127}-1$. The least period of the sequence should be a divisor of $2^{127}-1$. On the other hand, $2^{127}-1$ is a prime number (divided by 1 and itself). So the least period of $S''_0, S''_1, \ldots$ is $2^{127}-1$.

Next, assume that $S''_i = S''_j$ for i and j with $0 \leq i < j < 2^{127}-1$. Then $S'_{32i} = S'_{32j}$, which implies that $32(j-i)$ is a multiple of $2^{127}-1$. Since $2^{127}-1$ is prime to 32, $j-i$ is a multiple of $2^{127}-1$, which contradicts the assumption.

2.2 The Nonlinear Filter

The nonlinear filter is a memoryless function, that is, its output at time n only depends on the content of the word-oriented LFSR at time n. Let $(x_{n+3}, x_{n+2}, x_{n+1}, x_n)$ denote the content of the word-oriented LFSR at time n. The output at time n, denoted by z'_n, is described by the following pseudo-code:

```
Nonlinear-Function $F(x_{n+3}, x_{n+2}, x_{n+1}, x_n)$
1  $A \leftarrow x_{n+3} + (x_n \vee 1) \bmod 2^{32}$
2  $c \leftarrow$ carry
3  cyclic shift $A$ left 16 bits
4  **if** $(c = 0)$ **then**
5      $A \leftarrow A + x_{n+2} \bmod 2^{32}$
6  **else**
7      $A \leftarrow A + (x_{n+2} \oplus (x_n \vee 1)) \bmod 2^{32}$
8  $c \leftarrow$ carry
9  **return** A + $(x_{n+1} \oplus x_{n+2}) + c \bmod 2^{32}$
```

Let c_1 and c_2 denote the first (line 2) and second (line 8) carry bits in the pseudo-code. For a 32-bit integer A, let $\langle A \rangle_{16}$ denote the result of cyclicly shifting A left 16 bits. Then the function F has the following compact form:

$$z'_n = \langle x_{n+3} + (x_n \vee 1) \rangle_{16} + x_{n+2} \oplus c_1(x_n \vee 1) + x_{n+1} \oplus x_{n+2} + c_2 \bmod 2^{32}, \quad (4)$$

where $\vee$ denotes bitwise "OR" operation. The priority of $\oplus$ is assumed to be higher than that of $+$. Note that the least significant bit of x_n is always masked by 1 in order to get rid of the effect of stage 0 of the LFSR in Figure 2.

2.3 The Lagged-Fibonacci Generator

Lagged-Fibonacci generators, also called additive generators, have been widely used as random number generators in Monte Carlo simulation [4,6]. Mathematically, a lagged Fibonacci generator can be characterized by the following recursion:

$$y_n = y_{n-s} + y_{n-r} \bmod M, \quad n \geq r. \quad (5)$$

The generator is defined by the modulus M, the register length r, and the lag s, where $r > s$. When M is prime, periods as large as $M^r - 1$ can be achieved for the generated sequences. However it is more common to use lagged-Fibonacci generators with $M = 2^m$, $m \geq 1$. These generators with power-of-two moduli are much easier to implement than prime moduli. The following lemma was proved by Brent [1].

Lemma 1. Assume that $M = 2^m, m \geq 1, r > 2$, and the polynomial $x^r + x^s + 1$ is primitive over $GF(2)$. Then the sequence $y_0, y_1, \ldots$ of the lagged-Fibonacci generator described by (5) has the least period $2^{m-1}(2^r - 1)$ if $y_0, y_1, \ldots, y_{r-1}$ are not all even.

In SSC2, the lagged-Fibonacci generator with $s = 5$, $r = 17$, and $M = 2^{32}$ was adopted. We implemented this generator with a 17-stage circular buffer, B, and two pointers, s, and, r. Initially $\mathsf{B}[17], \mathsf{B}[16], \ldots, \mathsf{B}[1]$ are loaded with $y_0, y_1, \ldots, y_{16}$, and s and r are set to 5 and 17, respectively. At every clock, a new word is produced by taking the sum of $\mathsf{B}[\mathsf{r}]$ and $\mathsf{B}[\mathsf{s}]$ $\bmod 2^{32}$, the word $\mathsf{B}[\mathsf{r}]$ is then replaced by the new word, and the pointers s and r are decreased by 1. In this way, the buffer B produces the lagged-Fibonacci sequence. We use a multiplexer to generate the output sequence $z''_n, n \geq 0$ The output word z''_n is computed from the replaced word y_n and another word selected from the buffer B. The selection is based on the most significant 4 bits of the newly produced word y_{n+17}. The output word at time n, denoted by z''_n, is given by

$$z''_n = \langle y_n \rangle_{16} + \mathsf{B}[1 + ((y_{n+17} \gg_{28}) + \mathsf{s}_{n+1} \bmod 16)] \bmod 2^{32}, \quad (6)$$

where s_{n+1} denotes the value of s at time $n+1$. The pseudo-code for z''_n is listed as follows:

```
$A \leftarrow \mathsf{B}[\mathsf{r}]$
$D \leftarrow \mathsf{B}[\mathsf{s}] + \mathsf{B}[\mathsf{r}] \bmod 2^{32}$
$\mathsf{B}[\mathsf{r}] \leftarrow D$
$\mathsf{r} \leftarrow \mathsf{r} - 1$
$\mathsf{s} \leftarrow \mathsf{s} - 1$
**if** $(\mathsf{r} = 0)$ **then** $\mathsf{r} \leftarrow 17$
**if** $(\mathsf{s} = 0)$ **then** $\mathsf{s} \leftarrow 17$
cyclicly shift $A$ left 16 bits
**output** $A + \mathsf{B}[1 + (\mathsf{s} + D \gg_{28} \bmod 16)] \bmod 2^{32}$
```

Proposition 2. Assume that the initial state $y_0, y_1, \ldots, y_{16}$ of the lagged-Fibonacci generator are not all even. Then the sequence $\tilde{z}'' = z''_0, z''_1, \ldots$ is periodic and its least period is a divisor of $17(2^{17} - 1)2^{31}$.

Proof. Let $\mathsf{r_n}$ and $\mathsf{s_n}$ denote the values of r and s at time n. It is easy to verify that

$$\mathsf{r}_n = 17 - (n \bmod 17),$$

and

$$\mathsf{s}_n = 17 - (n + 12 \bmod 17).$$

Hence, the two sequences $\tilde{\mathsf{r}} = \mathsf{r}_0, \mathsf{r}_1, \ldots$ and $\tilde{\mathsf{s}} = \mathsf{s}_0, \mathsf{s}_1, \ldots$ are periodic and have the period 17. Since $y_0, y_1, \ldots, y_{16}$ are not all even, by Lemma 5.1, the lagged-Fibonacci sequence $\tilde{y} = y_0, y_1, \ldots$ is periodic and has the period $(2^{17} - 1)2^{31}$. Let $T_{\tilde{y}}$ denote the period of $\tilde{y}$. For any $1 \leq i \leq 17$, at time $n = 17T_{\tilde{y}} - i$, the pointer r has value $\mathsf{r}_n = 17 - (n \bmod 17) = i$, thus, the content of $\mathsf{B}[i]$ is replaced by $y_{n+17} = y_{17T_{\tilde{y}}-i+17} = y_{17-i}$. Hence, at time $n = 17T_{\tilde{y}} - 17$, the word in $\mathsf{B}[17]$ is replaced by y_0, at time $n = 17T_{\tilde{y}} - 16$, the word in $\mathsf{B}[16]$ is replaced by y_1, ..., at time $n = 17T_{\tilde{y}} - 1$, the word in $\mathsf{B}[1]$ is replaced by y_{16}. Therefore, at time $17T_{\tilde{y}}$, the content of B is the same as its content at time $n = 0$. Similarly, it can be proved that the content of B at time $n + 17T_{\tilde{y}}$ is the same as its content at time n. By (6), z''_n can be expressed by

$$z''_n = \langle y_n \rangle_{16} + \mathsf{B}[1 + ((y_{n+17} \gg 28) + \mathsf{s}_{n+1} \bmod 16)].$$

Let i_n denote the index in B in the above equation, namely,

$$i_n = 1 + ((y_{n+17} \gg 28) + \mathsf{s}_{n+1} \bmod 16).$$

Then

$$i_{n+17T_{\tilde{y}}} = 1 + ((y_{n+17T_{\tilde{y}}+17} \gg 28) + \mathsf{s}_{n+17T_{\tilde{y}}+1} \bmod 16 = i_n).$$

Consequently,

$$z''_{n+17T_{\tilde{y}}} = \langle y_{n+17T_{\tilde{y}}} \rangle_{16} + \mathsf{B}[i_{n+17T_{\tilde{y}}}] \bmod 2^{32} = z''_n,$$

which implies that the period of $\tilde{z}''$ divides $17T_{\tilde{y}}$.

3 Cryptographic Properties of SSC2

Period, linear complexity, and statistical distribution are three fundamental measures of security for keystream generators. Unfortunately, these measures are difficult to analyze for most of the proposed keystream generators. In this section, an assessment of the strength of SSC2 will be carried out with respect to these measures.

In the following, we will use $\tilde{z} = z_0, z_1, \ldots$ to denote the keystream sequence generated by SSC2. It is the sum of the two sequences $\tilde{z}' = z'_0, z'_1, \ldots$ of the filter generator, and $\tilde{z}'' = z''_0, z''_1, \ldots$ of the lagged-Fibonacci generator.

Theorem 1. *Assume that the initial state S''_0 of the word-oriented LFSR is not zero, and the the initial state $y_0, y_1, \ldots, y_{16}$ of the lagged-Fibonacci generator are not all even. Then the least period of the keystream sequence generated by SSC2 is greater than or equal to $2^{128} - 2$.*

Proof. Let $T_{\tilde{z}}, T_{\tilde{z}'}$, and $T_{\tilde{z}''}$ denote the least periods of $\tilde{z}, \tilde{z}'$, and $\tilde{z}''$, respectively. By Proposition 1 and Proposition 2, $T_{\tilde{z}'} = 2^{127} - 1$, and $T_{\tilde{z}''}$ is a factor of $17(2^{17} - 1)2^{31}$. Since $2^{127} - 1$ and $17(2^{17} - 1)2^{31}$, are relatively prime, $T_{\tilde{z}'}$ and $T_{\tilde{z}''}$ are also relatively prime. Hence $T_{\tilde{z}} = T_{\tilde{z}'}T_{\tilde{z}''}$. Therefore $T_{\tilde{z}} \geq 2T_{\tilde{z}'} = 2^{128} - 2$.

Let $\Lambda(\tilde{z}')$ and $\Lambda(\tilde{z}'')$ denote the linear complexity of $\tilde{z}'$ and $\tilde{z}''$. According to [11], the linear complexity of $\tilde{z} = \tilde{z}' \oplus \tilde{z}''$ is bounded by

$$\Lambda(\tilde{z}') + \Lambda(\tilde{z}'') - 2\gcd(T_{\tilde{z}'}, T_{\tilde{z}''}) \leq \Lambda(\tilde{z}) \leq \Lambda(\tilde{z}') + \Lambda(\tilde{z}''). \tag{7}$$

Thus, if we have lower bounds on the linear complexity of either $\tilde{z}'$ or $\tilde{z}''$, we can achieve lower bounds on the linear complexity of $\tilde{z}$. In the following, we will analyze the linear complexity of $\tilde{z}'$.

We can treat the sequence $\tilde{z}' = z'_0, z'_1, \ldots$ in three different forms. First, it is a sequence of words with $z'_n = (z'_{31,n}, z'_{30,n}, \ldots, z'_{0,n})$; second, it is a sequence of bits; and third, it can be considered as a collection of 32 component sequences, $\tilde{z}'_i = z'_{i,0}, z'_{i,1}, \ldots, 0 \leq i \leq 31$. For any $0 \leq i \leq 31$, $z'_{i,n}$ can be described by

$$z'_{i,n} = f_i(s''_{127,n}, s''_{126,n}, \ldots, s''_{1,n}), \quad n \geq 0, \tag{8}$$

where $(s''_{127,n}, s''_{126,n}, \ldots, s''_{1,n})$ is the state of the word-oriented LFSR at time n, and f_i is the i-th component of the nonlinear filter F. Assume that the nonlinear order, $ord(f_i)$, of f_i is ℓ_i. From Key's analysis [5], the linear complexity of $\tilde{z}'_i$ is bounded by

$$\Lambda(\tilde{z}'_i) \leq L_{\ell_i} = \sum_{j=1}^{\ell_i} \binom{127}{j}. \tag{9}$$

The upper bound L_{ℓ_i} is usually satisfied with equality. But, there are also few exceptions that the actual linear complexity is deviated from the expected value given by the upper bound. Rueppel [12] proved that, for a LFSR with primitive connection polynomial of prime degree L, the fraction of Boolean functions of nonlinear order ℓ that produce sequences of linear complexity L_ℓ is

$$P_d \approx \exp(-L_\ell/(L \cdot 2^L)) > e^{-1/L}. \tag{10}$$

For the filter generator of SSC2, we have $P_d > e^{-\frac{1}{127}}$. Hence, the linear complexity of $\tilde{z}_i'$ is virtually certain to be L_{ℓ_i}.To determine the nonlinear order of f_i, we will study the nonlinear order of integer addition.

Let $x = (x_{31}, x_{30}, \ldots, x_0)$ and $y = (y_{31}, y_{30}, \ldots, y_0)$ denote the binary representation of two 32-bit integers, where x_0 and y_0 are the least significant bits of x and y. The sum, $x + y \bmod 2^{32}$, defines a new 32-bit integer $z = (z_{31}, z_{30}, \ldots, z_0)$. The binary digits $z_i, 0 \le i \le 31$, are recursively computed by

$$z_i = x_i \oplus y_i \oplus c_{i-1}, \tag{11}$$

$$c_i = x_i y_i \oplus (x_i \oplus y_i) c_{i-1}, \tag{12}$$

where c_{i-1} denotes the carry bit, and $c_{-1} = 0$. The 31st carry bit c_{31} is also called the carry bit of $x + y$. In (13) and (14), z_i and c_i are Boolean functions of x_i and y_i, $0 \le i \le 31$. In the following, we will use $ord(z_i)$ and $ord(c_i)$ to denote the nonlinear order of the Boolean functions represented by z_i and c_i.

Lemma 2. *Assume that $x = (x_{31}, x_{30}, \ldots, x_0)$, $y = (y_{31}, y_{30}, \ldots y_0)$ are two 32-bit integers, and $x_0 = 1$. Let $z = (z_{31}, z_{30}, \ldots, z_0)$ denote the sum of $x + y$, and $c = (c_{31}, c_{30}, \ldots, c_0)$ denote the carry bits produced by the summation. Then $ord(c_i) = i + 1, 0 \le i \le 31$. Furthermore, $ord((x_i \oplus y_i)c_{31}) = 32$, $1 \le i \le 31$, and $ord(c_{15}c_{31}) = 33$.*

Proof. By (14), $c_0 = y_0$, and $c_1 = x_1 y_1 \oplus (x_1 \oplus y_1) y_0$. So $ord(c_0) = 1$ and $ord(c_1) = 2$. Assume that $ord(c_i) = i+1, i \ge 2$. Since x_{i+1} and y_{i+1} do not appear in the Boolean function represented by c_i, $ord(x_{i+1}y_{i+1}) \le ord((x_{i+1} \oplus y_{i+1})c_i)$ for $i \ge 2$,

$$\begin{aligned} ord(c_{i+1}) &= ord(x_{i+1}y_{i+1} \oplus (x_{i+1} \oplus y_{i+1})c_i) \\ &= ord((x_{i+1} \oplus y_{i+1})c_i) \\ &= i + 2 \end{aligned}$$

By induction, $ord(c_i) = i + 1, 0 \le i \le 31$. Using similar techniques, it can be proved that $ord((x_i \oplus y_i)c_{31}) = 32$ and $ord(c_{15}c_{31}) = 33$.

Lemma 3. *Let $z = F(x_3, x_2, x_1, x_0)$ denote the nonlinear function described by (4), where $z = (z_{31}, z_{30}, \ldots, z_0)$, and $x_i = (x_{i,31}, x_{i,30}, \ldots, x_{i,0}), 0 \le i \le 3$. For any $0 \le i \le 31$, let $z_i = f_i(x_3, x_2, x_1, x_0)$, then $ord(f_i) \ge 64 + i$.*

Proof. Recall that F is a mapping of $GF(2)^{128}$ to $GF(2)^{32}$ given by

$$z = \langle x_3 + (x_0 \vee 1) \rangle_{16} + x_2 \oplus c_1(x_0 \vee 1) + x_1 \oplus x_2 + c_2 \bmod 2^{32},$$

where c_1 and c_2 are the carry bits produced by the first two additions.

Let $Z_1 = (z_{1,31}, z_{1,30}, \ldots, z_{1,0})$ denote the sum of x_3 and $x_0 \vee 1$. The carry bits produced by the summation is denoted by $c_{1,31}, c_{1,30}, \ldots, c_{1,0}$. It is clear that $c_1 = c_{1,31}$. By Lemma 2, $ord(c_1) = 32$, and $ord(z_{1,i}) = ord(c_{1,i-1}) = i, 1 \leq i \leq 31$.

Let $Z'_1 = (z'_{1,31}, z'_{1,30}, \ldots, z'_{1,0})$ denote $\langle Z_1 \rangle_{16}$, that is, $z'_{1,i} = z_{1,16+i}$, for $0 \leq i \leq 15$, and $z'_{1,i} = z_{1,i-16}$, for $16 \leq i \leq 31$. Let $Z_2 = (z_{2,31}, z_{2,30}, \ldots, z_{2,0})$ denote the sum of Z'_1 and $x_2 \oplus c_1(x_0 \vee 1)$, and $c_{2,31}, c_{2,30}, \ldots, c_{2,0}$ denote the carry bits produced by the summation with $c_{2,-1} = c_1$. By (13) and (14),

$$z_{2,i} = z'_{1,i} \oplus x_{2,i} \oplus c_1 x_{0,i} \oplus c_{2,i-1} \tag{13}$$

$$c_{2,i} = z'_{1,i}(x_{2,i} \oplus c_1 x_{0,i}) \oplus (z'_{1,i} \oplus x_{2,i} \oplus c_1 x_{0,i})c_{2,i-1}, \tag{14}$$

where $c_{2,-1} = 0$ and $x_{0,0} = 1$. Rewriting (16), we have

$$c_{2,i} = (z'_{1,i} \oplus c_{2,i-1})x_{2,i} \oplus z'_{1,i}c_1 x_{0,i} \oplus (z'_{1,i} \oplus c_1 x_{0,i})c_{2,i-1}.$$

Since $x_{2,i}$ does not appear in $z'_{1,i}c_1 x_{0,i} \oplus (z'_{1,i} \oplus c_1 x_{0,i})c_{2,i-1}$, we have

$$ord(c_{2,i}) \geq ord((z'_{1,i} \oplus c_{2,i-1})x_{2,i}). \tag{15}$$

The carry bit $c_{2,0}$ has the following expression,

$$\begin{aligned} c_{2,0} &= z'_{1,0}(x_{2,0} \oplus c_1 x_{0,0}) \\ &= z_{1,16}(x_{2,0} \oplus c_{1,31}) \\ &= (x_{3,16} \oplus x_{0,16} \oplus c_{1,15})x_{2,0} \oplus (x_{3,16} \oplus x_{0,16})c_{1,31} \oplus c_{1,15}c_{1,31} \end{aligned}$$

By Lemma 2, $ord((x_{3,16} \oplus x_{0,16})c_{1,31}) = 32$, and $ord(c_{1,15}c_{1,31}) = 33$. The order of $(x_{3,16} \oplus x_{0,16} \oplus c_{1,15})x_{2,0}$ is equal to 17. So the order of $c_{2,0}$ equals 33. On the other hand, $ord(z_{1,i}) \leq i, 0 \leq i \leq 31$. Hence, $ord(c_{2,0}) > ord(z'_{1,1}) = ord(z_{1,17})$. By (17), $ord(c_{2,1}) \geq ord(c_{2,0}x_{2,1}) > 33$. By induction, it can be proved that $ord(c_{2,i}) \geq 33, 0 \leq i \leq 31$. Thus $ord(c_{2,i}) \geq ord(c_{2,i-1}) + 1$. Hence $ord(c_{2,31}) \geq 64$.

Let $Z_3 = (z_{3,31}, z_{3,30}, \ldots, z_{3,0})$ denote the sum of $Z_2 + x_1 \oplus x_2 + c_2$. The carry bits produced by the summation is denoted by $c_{3,31}, c_{3,30}, \ldots, c_{3,0}$. It is obvious that $c_2 = c_{2,31}$, and $z = Z_3$. By (13) and (14), we have the following expressions for z_i and $c_{3,i}$,

$$z_i = z_{2,i} \oplus x_{1,i} \oplus x_{2,i} \oplus c_{3,i-1} \tag{16}$$

$$c_{3,i} = z_{2,i}(x_{1,i} \oplus x_{2,i}) \oplus (z_{2,i} \oplus x_{1,i} \oplus x_{2,i})c_{3,i-1}, \tag{17}$$

where $c_{3,-1} = c_{2,31}$. By (15), $ord(z_{2,0}) = ord(c_1) = 32$. Thus, $ord(z_0) = ord(c_{3,-1}) \geq 64$. Rewriting (19), we have the following expression for $c_{3,i}$,

$$c_{3,i} = (z_{2,i} \oplus c_{3,i-1})x_{1,i} \oplus z_{2,i}x_{2,i} \oplus (z_{2,i} \oplus x_{2,i})c_{3,i-1}. \tag{18}$$

Substituting (20) into (18),

$$z_i = z_{2,i} \oplus x_{1,i} \oplus x_{2,i} \oplus (z_{2,i-1} \oplus c_{3,i-2})x_{1,i-1} \oplus z_{2,i-1}x_{2,i-1} \oplus (z_{2,i-1} \oplus x_{2,i-1})c_{3,i-2}. \tag{19}$$

Since $x_{1,i-1}$ only appears in $(z_{2,i-1} \oplus c_{3,i-2})x_{1,i-1}$, it is clear that

$$ord(z_i) \geq ord(z_{2,i-1} \oplus c_{3,i-2}) + 1. \tag{20}$$

By (20), $z_{2,i} \oplus c_{3,i-1}$ is described by

$$z_{2,i} \oplus c_{3,i-1} = z_{2,i} \oplus (z_{2,i-1} \oplus c_{3,i-2})x_{1,i-1} \oplus z_{2,i-1}x_{2.i-1} \oplus (z_{2,i-1} \oplus x_{2,i-1})c_{3,i-2}. \tag{21}$$

Since $x_{1,i-1}$ appears only in the second term $(z_{2,i-1} \oplus c_{3,i-2})x_{1,i-1}$, $ord(z_{2,i} \oplus c_{3,i-1}) \geq ord(z_{2,i-1} \oplus c_{3,i-2}) + 1$. Let $d_i = ord(z_{2,i} \oplus c_{3,i-1})$, $0 \leq i \leq 31$. Then $d_i \geq d_{i-1} + 1$. On the other hand,

$$\begin{aligned} d_0 &= ord(z_{2,0} \oplus c_{3,-1}) \\ &= ord(z_{2,0} \oplus c_{2,31}) \\ &\geq 64. \end{aligned}$$

Hence, $d_i \geq 64 + i$, $1 \leq i \leq 31$. By (22), $ord(z_i) \geq 64 + i$, which proves the lemma.

Theorem 2. Let $\tilde{z} = z_0, z_1, \ldots$ denote the word sequence generated by SSC2. For any $n \geq 0$, let $z_n = (z_{31,n}, z_{30,n}, \ldots, z_{0,n})$. Let S_0'' be the initial state of the word-oriented LFSR and $y_0, y_1, \ldots, y_{16}$ be the initial state of the lagged-Fibonacci generator. Assume that S_0'' is not zero and $y_0, y_1, \ldots, y_{16}$ are not all even. Then, with a probability greater than $e^{-\frac{1}{127}}$, the binary sequences $\tilde{z}_i = z_{i,0}z_{i,1}\ldots, 0 \leq i \leq 31$, have linear complexity

$$\Lambda(\tilde{z}_i) \geq \sum_{j=1}^{64+i} \binom{127}{j} - 2 \geq 2^{126}.$$

Proof. Similar to the decomposition of $\tilde{z} = z_0, z_1, \ldots$, we can decompose the word sequence $\tilde{z}' = z_0', z_1', \ldots$ generated by the filter generator into 32 component sequences, $\tilde{z}_i' = z_{i,0}', z_{i,1}', \ldots, 0 \leq i \leq 31$. Similarly, let $\tilde{z}_i'' = z_{i,0}'', z_{i,1}'', \ldots, 0 \leq i \leq 31$ denote the component sequences of $\tilde{z}'' = z_0'', z_1'', \ldots$ generated by the lagged-Fibonacci generator. Then $\tilde{z}_i = \tilde{z}_i' \oplus z_i''$. Let $T_{\tilde{z}_i}$,$T_{\tilde{z}_i'}$, and $T_{\tilde{z}_i''}$ denote the least period of $\tilde{z}_i$, $\tilde{z}_i'$, and $\tilde{z}_i''$. By Proposition 1, the word sequence $\tilde{z}' = z_0', z_1', \ldots$ has the least period $2^{127} - 1$. Hence, the component sequence $\tilde{z}_i'$ has a period of $(2^{127} - 1)$. By Proposition 2, it is easy to verify that the sequence $\tilde{z}_i''$ has a period of $17(2^{17} - 1)2^{31}$. Therefore, $\gcd(T_{\tilde{z}_i'}, T_{\tilde{z}_i''}) = 1$. By (7), we have

$$\Lambda(\tilde{z}_i) \geq \Lambda(\tilde{z}_i') - 2$$

By Lemma 3, with a probability no less than $e^{-\frac{1}{127}}$, the linear complexity of $\tilde{z}_i'$ is at least L_{64+i}, which proves the theorem.

Theorem 2 implies that the linear complexity of the component sequences $\tilde{z}_i, 0 \leq i \leq 31$, is exponential to the length of the LFSR and is therefore, resilient to the Berlekamp-Massey attack.

We next study the question concerning how close a SSC2 sequence resembles a truly random sequence. Mathematically, a truly random sequence can be modeled as a sequence of independent and uniformly distributed random variables. To measure the randomness of the keystream sequences generated by SSC2, let's consider the distribution of every 32-bit word in a period.

Proposition 3. Let X_3, X_2, X_1, and X_0 be independent and uniformly random variables over $GF(2)^{32}$ and $Z = F(X_3, X_2, X_1, X_0)$ be the output of the filter generator in SSC2. Then Z is uniformly distributed over $GF(2)^{32}$.

Proof. Let $Z' = \langle X_3 + (X_0 \vee 1)\rangle_{16} + X_2 \oplus c_1(X_0 \vee 1) + c_2 \bmod 2^{32}$. Then $Z = X_1 \oplus X_2 + Z' \bmod 2^{32}$. By the chain rule [2], we can express the joint entropy $H(Z, Z', X_1 \oplus X_2)$ as

$$\begin{aligned} H(Z, Z', X_1 \oplus X_2) &= H(Z') + H(Z|Z') + H(X_1 \oplus X_2|Z, Z') \\ &= H(Z') + H(X_1 \oplus X_2|Z') + H(Z|X_1 \oplus X_2, Z'). \end{aligned}$$

Since $X_1 \oplus X_2$ is uniquely determined by Z and Z', $H(X_1 \oplus X_2|Z, Z') = 0$. Similarly, $H(Z|X_1 \oplus X_2, Z') = 0$. Hence, $H(Z|Z') = H(X_1 \oplus X_2|Z')$.

Since X_1 does not appear in the expression represented by Z', X_1 and Z' are statistically independent. For any $a, b \in GF(2)^{32}$,

$$\begin{aligned} p(X_1 \oplus X_2 = a|_{Z'=b}) &= \frac{p(X_1 \oplus X_2 = a, Z' = b)}{p(Z' = b)} \\ &= \frac{\sum_{c \in GF(2)^{32}} p(X_1 \oplus X_2 = a|_{Z'=b, X_2=c}) p(Z' = b, X_2 = c)}{p(Z' = b)} \\ &= \frac{\sum_{c \in GF(2)^{32}} p(X_1 = a \oplus c|_{Z'=b, X_2=c}) p(Z' = b, X_2 = c)}{p(Z' = b)}. \end{aligned}$$

Since X_1 and (Z', X_2) are independent, $p(X_1 = c \oplus a|_{Z'=b, X_2=c}) = 2^{-32}$. Thus,

$$p(X_1 \oplus X_2 = a|_{Z'=b}) = \frac{2^{-32} \sum_{c \in GF(2)^{32}} p(Z' = b, X_2 = c)}{p(Z' = b)} = 2^{-32}.$$

Hence, $H(Z|Z') = H(X_1 \oplus X_2|Z') = 32$. Therefore, $H(Z) \geq H(Z|Z') = 32$, which implies that $H(Z) = 32$, or equivalently, Z is uniformly distributed over $GF(2)^{32}$.

Recall that the state sequence $S''_0, S''_1, \ldots$ of the word-oriented LFSR has the least period $2^{127} - 1$ and all states are distinct in the period if the initial state S''_0 is non-zero. Hence every non-zero state appears exactly once in the least period. For this reason, we model the state S''_n as a uniformly distributed random variable over $GF(2)^{127}$. Proposition 3 indicates that we can model the filter generator sequence as a sequence of uniformly distributed random variables when the initial state of the filter generator is non-zero.

Theorem 3. Let S_0'' and $Y_0 = (y_0, y_1, \ldots, y_{16})$ be the initial states of the LFSR and the lagged-Fibonacci generator respectively. Assume that S_0'' and Y_0 are random variables, and $S_0'' \neq 0$. Let Z_n denote the word generated by SSC2 at time n. Then

$$32 - I(S_0''; Y_0) \leq H(Z_n) \leq 32,$$

where $I(S_0''; Y_0)$ denotes the mutual information between S_0'' and Y_0, given by

$$I(S_0''; Y_0) = H(S_0'') - H(S_0''|Y_0).$$

Proof. Since Z_n is a random variable over $GF(2)^{32}$, it is obvious that $H(Z_n) \leq 32$. Let Z_n' and Z_n'' denote the respective output of the filter generator and the lagged-Fibonacci generator at time n. Then $Z_n = Z_n' + Z_n'' \bmod 2^{32}$. Moreover, $H(Z_n|Z_n'') = H(Z_n'|Z_n'')$. According to the data processing inequality [2],

$$I(Z_n'; Z_n'') \leq I(Z_n'; Y_0) \leq I(S_0''; Y_0).$$

Since $S_0'' \neq 0$, $H(Z_n') \approx 32$. Consequently,

$$\begin{aligned} H(Z_n) &\geq H(Z_n|Z_n'') \\ &= H(Z_n'|Z_n'') \\ &= H(Z_n') - I(Z_n'; Z_n'') \\ &\geq 32 - I(S_0''; Y_0). \end{aligned}$$

By Theorem 3, we can conclude that the keystream sequence of SSC2 is a sequence of uniformly distributed random variables if the initial states of the word-oriented LFSR is non-zero and statistically independent of the initial state of the lagged-Fibonacci generator. However, the problem of determining whether the keystream sequence of SSC2 is a sequence of independent random variables or not remains open.

4 Correlation Analysis of SSC2

SSC2 is a very complex mathematical system in which several different types of operations, such as exclusive-or, integer addition, shift, and multiplexing, are applied to data iteratively. If we analyze the keystream generator in Figure 1 as a whole, it would be difficult to get information about the internal states of the word-oriented LFSR and the lagged-Fibonacci generator. However, if the keystream sequence leaks information about the filter generator sequence or the lagged-Fibonacci sequence, this information might be exploited to attack the filter generator or the lagged-Fibonacci generator separately. This kind of attack is called divide-and-conquer correlation attack which has been successfully applied to over a dozen keystream generators [3,8,10,11,14,15]. For the moment, let's assume that the key of SSC2 consists of the initial states S_0'' and Y_0 of the word-oriented LFSR and the lagged-Fibonacci generator respectively.

Theorem 4. Assume that the initial states S_0'' and Y_0 of the filter generator and the lagged-Fibonacci generator of SSC2 are random variables, $S_0'' \neq 0$. Then the outputs Z_n' and Z_n'' of the filter generator and the lagged-Fibonacci generator at time n are also random variables. Let Z_n denote the output of SSC2 at time n. Then

$$\begin{aligned} I(Z_n; Y_0) &\leq 32 - H(Z_n') + I(S_0''; Y_0), \\ I(Z_n; S_0'') &\leq 32 - H(Z_n'') + I(S_0''; Y_0). \end{aligned}$$

Proof. By the chain rule [2], the joint entropy $H(Z_n', Y_0, Z_n)$ can be represented as follows:

$$\begin{aligned} H(Z_n', Y_0, Z_n) &= H(Y_0) + H(Z_n|Y_0) + H(Z_n'|Z_n, Y_0) \\ &= H(Y_0) + H(Z_n'|Y_0) + H(Z_n|Z_n', Y_0). \end{aligned}$$

Since Z_n' can be uniquely determined by Z_n and Y_0, $H(Z_n'|Z_n, Y_0) = 0$. Similarly, $H(Z_n|Z_n', Y_0) = 0$. Thus, $H(Z_n|Y_0) = H(Z_n'|Y_0)$. Therefore,

$$\begin{aligned} I(Z_n; Y_0) &= H(Z_n) - H(Z_n|Y_0) \\ &= H(Z_n) - H(Z_n'|Y_0) \\ &= H(Z_n) - H(Z_n') + I(Z_n'; Y_0) \end{aligned}$$

According to the data processing inequality [2], $I(Z_n'; Y_0) \leq I(S_0''; Y_0)$. Hence, it follows that

$$I(Z_n; Y_0) \leq H(Z_n) - H(Z_n') + I(S_0''; Y_0).$$

On the other hand, $H(Z_n) \leq 32$,

$$I(Z_n; Y_0) \leq 32 - H(Z_n') + I(S_0''; Y_0).$$

Similarly, it can be proved that

$$I(Z_n; S_0'') \leq 32 - H(Z_n'') + I(S_0''; Y_0).$$

According to the empirical test in [7], we assume that the sequence $\tilde{y} = y_0, y_1, \ldots$ of the lagged-Fibonacci generator is a sequence of pairwise independent and uniformly distributed random variables. By (6), $Z_n'' = \langle y_n \rangle_{16} + y_n' \bmod 2^{32}$, where y_n' is uniformly selected from $y_{n+1}, y_{n+2}, \ldots, y_{n+17}$. If $y_n' \neq y_{n+17}$, it is obvious that Z_n'' is uniformly distributed. If $y_n' = y_{n+17}$, then $Z_n'' = \langle y_n \rangle_{16} + y_n + y_{n+12} \bmod 2^{32}$, which is also uniformly distributed since y_{n+12} and y_n are independent. Thus, for any $n \geq 0$, Z_n is not correlated to either S_0'' or Y_0 if S_0'' and Y_0 are statistically independent. So we can not get any information about S_0'' or Y_0 from each Z_n. However, this does not mean that we can not get information about S_0'' and Y_0 from a segment $Z_0, Z_1, \ldots, Z_m$ of the keystream sequence. The question is how to get information about S_0'' and Y_0 from a segment of the keystream sequence, which remains open.

5 Scalability of SSC2

The security level of SSC2 can be enhanced by increasing the length of the lagged-Fibonacci generator. Let $\tilde{y} = y_0, y_1, \ldots$ be the word sequence generated by a lagged-Fibonacci generator of length L. As described in Section 2.3, we can implement the lagged-Fibonacci generator with a buffer B of length L and two pointers r and s. Let $h = \lfloor \log L \rfloor$. The output word z''_n of the lagged-Fibonacci generator can be described by

$$z''_n = y_n + \mathsf{B}[1 + ((y_{n+L} \gg (32 - h)) + \mathsf{s}_{n+1} \bmod 2^h)] \bmod 2^{32}, \qquad (22)$$

where s_{n+1} is the value of the pointer s at time $n+1$. We define the number $Lh = L\lfloor \log L \rfloor$ as the effective key length of the keystream generator as described by Figure 1, where the lagged-Fibonacci generator has length L. The effective key length gives us a rough estimation of the strength of the keystream generator. We believe that the actual strength might be much larger than that described by the effective length. Corresponding to private keys of 128 bits, lagged-Fibonacci generators with length between 17 and 33 are recommended.

6 Key Scheduling Scheme

SSC2 supports private keys of various sizes, from 4 bytes to 16 bytes. To stretch a private key less than or equal to 4 words to 21 words, a key scheduling scheme is required. By Theorem 3 and Theorem 4, the initial states of the word-oriented LFSR and the lagged-Fibonacci generator should be independent. With a hash function such as SHA-1[9], it is not difficult to generate such 21 words. When a good hash function is not available, we designed the following scheme which generates 21 words (called the master key) from the private key K.

```
Master-Key-Generation $K_{master}(K)$
1   load $K$ into the LFSR $S$, repeat $K$ when necessary
2   **for** $i \leftarrow 0$ **to** 127 **do**
3       run the linear feedback shift register once
4       $S[1] \leftarrow S[1] + F(S) \bmod 2^{32}$
5       $i \leftarrow i + 1$
6   **for** $i \leftarrow 1$ **to** 17 **do**
7       run the linear feedback shift register once
8       $\mathsf{B}[i] \leftarrow S[4]$
9       $i \leftarrow i + 1$
10  $A \leftarrow S[1]$
11  **for** $i \leftarrow 1$ **to** 34 **do**
12      run the linear feedback shift register once
13      run the lagged Fibonacci generator once
14      $index \leftarrow 1 + A \gg 28$
15      $A \leftarrow \mathsf{B}[index]$
16      $\mathsf{B}[index] \leftarrow A \oplus S[1]$
```

```
17         $S[1] \leftarrow A + S[1] \bmod 2^{32}$
18         $i \leftarrow i + 1$
19     $\mathsf{B}[17] \leftarrow \mathsf{B}[17] \vee 1$
20     **return** $S$ and $\mathsf{B}$
```

In the above pseudo-code, the LFSR is denoted by S, which is an array of 4 words. Every time when the LFSR runs, the word in $S[4]$ moves out, the array shifts right one word, and the newly computed word moves in $S[1]$. The lagged-Fibonacci generator is denoted by B as usual. The master key generation experiences 5 stages. In stage 1, the private key is loaded into the LFSR. In stage 2 (line 2 - line 5) , the private key is processed. We run the LFSR (actually the filter generator) 128 times in order that approximately half of the bits in S will be 1 even if there is only one 1 in K. In stage 3 (line 6 - line 9), 17 words are generated for the lagged-Fibonacci generator. In stage 4 (line 10 - line 18) , the LFSR and the lagged-Fibonacci generator interact with each other for 34 times. A major goal for the interaction is to make it difficult to gain information about the state of the LFSR from the state of the lagged-Fibonacci generator and vice versa. For this purpose, an index register A is introduced, which has $S[1]$ as the initial value. At the end of each run of the LFSR and the lagged-Fibonacci generator run, a pointer *index* is computed according to the most significant 4 bits of A, and then A is updated by the word $\mathsf{B}[index]$. Following the update of A, $\mathsf{B}[index]$ is updated by $A \oplus S[1]$ and $S[1]$ is updated by $A+S[1] \bmod 2^{32}$. Through the register A, the states of the LFSR and the lagged-Fibonacci generator are not only related to each other but are also related to their previous states. For example, assume that the state of the LFSR is known at the end of stage 4. To obtain the previous state of the LFSR, we have to know the content of A, which is derived from the previous state of the lagged-Fibonacci generator. In stage 5, the least significant bit of $\mathsf{B}[17]$ is set to 1 in order to ensure that not all of the 17 words of B are even. At the end of the computation, the states of the LFSR and the lagged-Fibonacci generator are output as the master key.

In addition to the master key generation, SSC2 supplies an optional service of generating a key for every frame. The key for a frame is used to re-load the LFSR and the lagged-Fibonacci generator when the frame is encrypted. The purpose of frame-key generation is to cope with the synchronization problem. In wireless communications, there is a high probability that packets may be lost due to noise, or synchronization between the mobile station and the base station may be lost due to signal reflection, or a call might be handed off to a different base station as the mobile station roams. When frames are encrypted with their individual keys, the loss of a frame will not affect the decryption of subsequent frames.

Assume that each frame is labeled by a 32-bit frame number that is not encrypted. Let K_n denote the frame key of the n-th frame. The frame key generation should satisfy two fundamental requirements: (1) it is fast; and (2) it is difficult to gain information about K_i from K_j when $i \neq j$. Taking into consideration of the two requirements, we design a scheme that generates K_n from the master key K_{master} and the frame number n. To generate different keys for different frames, we divide the 32-bit frame number into 8 consecutive blocks and have each block involve in the frame key generation. Let $n_0, n_1, \ldots, n_7$ denote the 8 blocks of n, where n_0 is the least significant 4 bits of n and n_7 is the most significant 4 bits of n. The frame key generation is illustrated by the following pseudo-code:

```
Frame-Key-Generation $K_n(K_{master})$
1   load $K_{master}$ into $S$ and $\mathsf{B}$
2   for $j \leftarrow 0$ to 3 do
3       for $i \leftarrow 0$ to 7 do
4           $S[1] \leftarrow S[1] + \mathsf{B}[1 + (i + n_i \bmod 16)] \bmod 2^{32}$
5           $S[2] \leftarrow S[2] + \mathsf{B}[1 + (8 + i + n_i \bmod 16)] \bmod 2^{32}$
6           run the linear feedback shift register once
7           $\mathsf{B}[17 - (i + 8j \bmod 16)] \leftarrow S[1] \oplus \mathsf{B}[17 - (i + 8j \bmod 16)]$
8           $i \leftarrow i + 1$
9       $j \leftarrow j + 1$
10  $\mathsf{B}[17] \leftarrow \mathsf{B}[17] \vee 1$
11  return $S$ and $\mathsf{B}$
```

The frame key generation consists of two loops. Corresponding to each $n_i, 0 \leq i \leq 7$, the inner-loop (line 4 - line 8) selects two words from the buffer B to update the contents of $S[1]$ and $S[2]$. Then the LFSR in executed and the output word is used to update one word of B. The outer-loop executes the inner-loop 4 times. Assume that n and n' are two different frame numbers. After the first run of the inner-loop, some words in S and B will be different for n and n'. Subsequent runs are used to produce more distinct words in S and B.

Table 1. Throughput of SSC2

Machine	Size	Clock rate (MHz)	Memory (Mbyte)	OS	Compiler	Throughput (Mbits/s)
Sun SPARC2	32	40	30	Sun OS	gcc -O3	22
Sun Ultra 1	32	143	126	Sun Solaris	gcc -O3	143
PC	16	233	96	Linux	gcc -O3	118

7 Performance

We have run SSC2 on various platforms. Table 4.1 illustrates the experimental results derived from running the ANSI C code listed in Appendix 1. Key setup times are not included in Table 1. On a 16-bit processor (233MHz cpu), the time for the master key generation is approximately equal to the encryption time for one CDMA frame (384 bits in 20 ms duration), and the time for the frame key generation is about one-twentieth of the encryption time for one CDMA frame. Suppose that the CDMA phones have the 16-bit processor and the average conversion takes 3 minutes. Then the total time for frame key generations is about 2 ms. Hence, the overhead introduced by key setup is nearly negligible.

8 Conclusion

SSC2 is a fast software stream cipher portable on 8-, 16-, and 32-bit processors. All operations in SSC2 are word-oriented, no complex operations such as multiplication, division, and exponentiation are involved. SSC2 has a very compact structure, it can be easily remembered. SSC2 does not use any look-up tables and does not need any pre-computations. Its software implementation requires very small memory usage. SSC2 supports variable private key sizes, it has an efficient key scheduling scheme and an optional frame key scheduling scheme. Its keystream sequence has large period, large linear complexity and small correlation to the component sequences. SSC2 is one of the few software stream ciphers whose major cryptographic properties have been established.

Acknowledgments. The authors would like to thank Dr. Daniel Bleichenbacher of Bell Laboratories for pointing out a weakness in the early design of SSC2. The first author also likes to thank Dr. Burton Kaliski Jr. of RSA Laboragtories for his encouragement and valuable suggestions.

References

1. R. P. Brent,"On the periods of generalized Fibonacci recurrences", *Mathematics of Computation*, vol. 63, pp. 389-401, 1994.
2. T. M. Cover and Y. A. Thomas, *Elements of Information Theory*, John Wiley, 1991.
3. R. Forre, "A fast correlation attack on nonlinearly feedforward filtered shift register sequences", *Advances in Cryptology-Proceedings of EUROCRYPT'89 (LNCS 434)*, 586-595, 1990.
4. F. James, "A review of pseudo-random number generators", *Computer Physics Communications*, vol. 60, pp. 329-344, 1990.
5. E. L. Key, "An analysis of the structure and complexity of nonlinear binary sequence generators", *IEEE Transactions on Infor. Theory*, vol. 22, Nov. 1976.
6. D. E. Knuth, *The Art of Computer programming. Volume 2: Seminumerical Algorithms*, 3rd Edition, Addison-Wesley, 1997.
7. G. Marsaglia, "A current view of random number generators", *Computer Science and Statistics: Proc. 16th Symposium on the Interface*, Elsvier Science Publishers B. V. (North-Holland), 1985.
8. W. Meier and O. Staffelbach, "Fast correlation attacks on stream ciphers", *Journal of Cryptology*, vol. 1, no. 3, 159-176, 1989.
9. FIPS 180, "Secure hash standard", *Federal Information Processing Standard Publication* 180, April, 1995.
10. M.J.B. Robshow, "Stream ciphers", Technical Report TR-701 (version 2.0), RSA Laboratories, 1995.

11. R. Rueppel, "Stream Ciphers" in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, 1992, pp. 65-134.
12. R. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, Berlin, 1986.
13. B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, New York, 2nd edition, 1996.
14. T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only", *IEEE Transactions on Computing*, vol. 34, 1985, 81-85.
15. T. Siegenthaler, "Cryptanalyst's representation of nonlinearity filtered ml-sequences", *Advances in Cryptology-Proceedings of Eurocrypt'85 (LNCS)*, Springer-Verlag, 1986.
16. A. Biryukov and A. Shamir, "Real time cryptanalysis of the alleged A5/1 on a PC", in this Proceedinds.

Appendix 1

The following is the ANSI C code for the keystream generator of SSC2. The code for key setup is not included.

```
unsigned long int R1, R2, R3, R4, B[18], output, temp1, temp2;
int c, s=5, r=17;

temp1 = R2 ^ (R3<<31) ^ (R4>>1);
R4 = R3;
R3 = R2;
R2 = R1;
R1 = temp1;

temp1 = B[r];
temp2 = B[s] + temp1;
B[r] = temp2;
if (--r == 0) r = 17;
if (--s == 0) s = 17;
output = ((temp1>>16) ^ (temp1<<16))+B[(((temp2>>28)+s) & 0xf)+1];

temp1 = (R4 | 0x1) + R1;
c = (temp1 < R1);
temp2 = (temp1<<16) ^ (temp1>>16);
if (c) {
   temp1 = (R2  ^ (R4 | 0x1)) + temp2;
} else {
   temp1 = R2 + temp2; }
c = (temp1 < temp2);
output = (c + (R3 ^ R2) + temp1) ^ output;
```

Mercy: A Fast Large Block Cipher for Disk Sector Encryption

Paul Crowley

DataCash Ltd.
mercy@paul.cluefactory.org.uk

Abstract. We discuss the special requirements imposed on the underlying cipher of systems which encrypt each sector of a disk partition independently, and demonstrate a certificational weakness in some existing block ciphers including Bellare and Rogaway's 1999 proposal, proposing a new quantitative measure of avalanche. To address these needs, we present Mercy, a new block cipher accepting large (4096-bit) blocks, which uses a key-dependent state machine to build a bijective F function for a Feistel cipher. Mercy achieves 9 cycles/byte on a Pentium compatible processor.

Keywords: disk sector, large block, state machine, avalanche, Feistel cipher.
Mercy home page: *http://www.cluefactory.org.uk/paul/mercy/*

1 Introduction

Disk sector encryption is an attractive approach to filesystem confidentiality. Filesystems access hard drive partitions at the granularity of the sector (or block) where a sector is typically 4096 bits: read and write requests are expressed in sector numbers, and data is read and modified a sector at at a time. Disk sector encryption systems present a "virtual partition" to the filesystem, mapping each sector of the virtual partition to the corresponding sector, through an encrypting transformation, on a physical disk partition with the same disk geometry. The performance is typically better than file-level encryption schemes, since every logical sector read or write results in exactly one physical sector read or write, and confidentiality is also better: not only are file contents obscured, but also filenames, file sizes, directory structure and modification dates. These schemes are also flexible since they make no special assumptions about the way the filesystem stores the file data; they work equally well with raw database partitions as with filesystems, and can be transparently layered underneath disk caching and disk compression schemes. Linux provides some support for such filesystems through the "/dev/loop0" filesystem device.

The stream cipher SEAL [17] is well suited to this need. SEAL provides a strong cryptographic PRNG (CPRNG) whose output is seekable. Thus the entire disk can be treated as a single contiguous array of bytes and XORred with the output from the CPRNG; when making reads or writes of specific sectors the appropriate portion of the output can be generated without the need to generate the preceding bytes. The same effect can be achieved, somewhat less efficiently, by keying a CPRNG such as ARCFOUR [10] with a (key, sector number) pair and generating 512 bytes with which to encrypt the sector. These schemes are highly efficient and provide good security against an attacker who seizes an encrypted hard drive and attempts to gain information about its contents.

However, this is not strong against attackers with other channels open to them. They may have user privileges on the system they're trying to attack, and be able to access the ciphertext stored on the hard drive at times when it's shut down. Or they may try to modify sectors with known contents while carrying a drive from place to place. They may even be able to place hardware probes on the drive chain while logged in as a normal user, and sniff or modify ciphertext. Against these attacks, SEAL and ARCFOUR (used as described) are ineffective. For example, an attacker can write a large file of all zeroes and thereby find the fixed encryption stream associated with many sectors; once the file is deleted, the sectors might be re-used by other users with secure data to write, and this data is easily decrypted by XORing with the known stream. Or, if attackers can make a guess of the plaintext in a given sector, they can modify this to another plaintext of their choosing while they have access to the drive by XORing the ciphertext with the XOR difference between the two plaintexts.

File-based encryption schemes defeat these attacks by using a new random IV for each new plaintext and authenticating with a MAC. However, applying these techniques directly to sector encryption would require that the ciphertext for each sector be larger than the plaintext, typically by at least 64 bytes. Thus either the plaintext sectors would need to be slightly smaller than the natural hardware sector size, harming performance when mapping files into memory (and necessitating a thorough re-engineering of the filesystem code) or auxiliary information would have to be stored in other sectors, potentially adding a seek to each read and write. In either case the size overhead will be about 1.5 - 3.1%. It's worth investigating what can be achieved without incurring these penalties.

SFS [9] uses a keyless mixing transformation on the plaintext before applying a block chaining stream cipher. This greatly reduces the practical usefulness of many such attacks, but it falls short of the highest security that pure disk sector encryption systems can aspire to: that the mapping between each virtual and physical disk sector appears to be an independent random permutation to an attacker who expends insufficient computation to exhaustively search the keyspace. In other words, the theoretical best solution under these constraints is a strong randomised large block cipher.

Several proposals exist for building large block ciphers from standard cryptographic components such as hash functions and stream ciphers [1,11,2]; however, these are not randomised ciphers, and as Section 2 shows, they have certificational weaknesses. More seriously, no proposal comes close to offering the performance needed: bit rates equal to or better than disk transfer rates. Since small improvements in disk access efficiency can mean big improvements to every part of the user experience, and since performance considerations are one of the main reasons why filesystem encryption is not more widely used, it seems worthwhile to develop a new cipher designed specifically to meet this need.

The rest of this paper is organised as follows. Section 2 describes a certificational weakness applicable to several existing classes of large block cipher, and proposes a quantitiative measure of avalanche based on the attack. Section 3 lays out the design goals for the solution here, a new block cipher Mercy with a native block size of 4096 bits, and Section 4 describes the cipher itself. Section 5 discusses the design of the cipher in detail with reference to the performance and security goals of Section 3. Finally Section 6 discusses some of the lessons learned in the design of Mercy.

2 Avalanche and Certificational Weaknesses

[14] presents an attack on bidirectional MACs based on inducing collisions in the internal state of the MAC. This attack can be extended to show a certificational weakness in some large block cipher proposals. Note that neither keys nor plaintext can be recovered using this attack; it merely serves to distinguish the cipher from a random permutation.

In general form, the attack proceeds as follows. The bits in the plaintext are divided into two categories, "fixed" and "changing"; a selection of the bits of the ciphertext are chosen as "target" bits. A number of chosen plaintexts are encrypted, all with the same fixed bits and with changing bits chosen at random; the attack is a success if a collision in the target bits of the ciphertext is generated. Let w_k be the length of the key, w_t the number of target bits, and 2^{w_p} be the number of different plaintexts encrypted: if the following conditions are met:

- the result is statistically significant, ie the probability of seeing such a collision under these circumstances from a genuine PRP (approximately $2^{2w_p-w_t-1}$) is small; and
- $w_p < w_k - 1$, ie the work factor for the attack is less than that for a key guessing attack against the cipher

then a certificational weakness has been demonstrated. The attack works by inducing an internal collision in the data path from the changing bits to the fixed bits; the width of this path determines the number of plaintexts needed and thus the smallest w_p for which the attack can work provides a useful quantitative measure of avalanche. This attack can easily be converted to use the memory-efficient parallel collision finding techniques of [20], so memory usage does not present a serious obstacle to the practicality of the attack if 2^{w_p} adaptive chosen plaintexts can be encrypted.

This attack may be applied to [2] by choosing the first two blocks of the plaintext as the "changing" bits, and all of the output except the second two blocks as the "target" bits. If the blocksize of the underlying cipher is 64 bits, then 2^{33} chosen plaintexts should be sufficient to induce a collision in σ, resulting in a collision in all the target bits as desired; if it is 128 bits, 2^{65} will be needed.

This attack also extends to bidirectional chaining systems, in which the plaintext is encrypted first forwards and then backwards using a standard block cipher in a standard chaining mode; in this case, the first two blocks are the changing bits, all of the ciphertext except the first two blocks are the target bits, and the number of plaintexts required are as before; the collision is induced in the chaining state after the first two blocks. If the chaining mode is CBC or CFB, all of the output except the first block will be target bits, since a collision in the internal state after the second block means that the second block of ciphertext is identical.

Note that this attack is not applicable to any of the proposals in [1]; neither BEAR nor LION claim to be resistant to any kind of chosen plaintext attack, while LIONESS carries 256 or 320 bits of data between the two halves (depending on the underlying hash function), which would require 2^{129} or 2^{161} chosen plaintexts; this is outside the security goals of the cipher. However, it can be applied to BEAST from [11] by inducing a collision in the SHA-1 hash of R^{**} with 2^{81} chosen plaintexts; the changing bits are the first 160 bits of R^{**}, and the target bits are all of the ciphertext except the first 160 bits of T^{**}. This attack is clearly impractical at the moment but it violates our expectation that the cheapest way to distinguish a block cipher from a random permutation should be a brute force key guessing attack.

3 Mercy Design Goals

Mercy is a new randomised block cipher accepting a 4096-bit block, designed specifically for the needs of disk sector encryption; it achieves significantly higher performance than any large block cipher built using another cipher as a primitive, or indeed than any block cipher that I know of large or small.

It accepts a 128-bit randomiser; it is expected that the sector number will be used directly for this purpose, and therefore that most of the randomiser bits will usually be zero. This is also known as a "diversification parameter" in the terminology of [6], or "spice" in that of [19]. This last term avoids the misleading suggestion that this parameter might be random and is convenient for constructions such as "spice scheduling" and "spice material" and is used henceforth.

Mercy's keyschedule is based on a CPRNG; the sample implementation uses [10]. Though [10] takes a variable length key, Mercy does not aspire to better security than a cipher with a fixed 128-bit key size, so it's convenient for the purposes of specifying these goals to assume that the key is always exactly 128 bits.

– **Security**: Any procedure for distinguishing Mercy encryption from a sequence of 2^{128} independent random permutations (for the 2^{128} possible spices) should show no more bias towards correctness than a key guessing attack with the same work factor. However, we do not claim that ignorance of the spice used would make any attack harder; it's not intended that the spice

be hidden from attackers. For this reason, Mercy is not intended to be a K-secure or hermetic cipher in the terminology of [7].
- **Resistance to specific attacks**: Mercy is designed to be resistant in particular to linear and differential attacks, as well as to avoid the certificational weakness of Section 2.
- **Speed**: Encryption and decryption should be much faster than disk transfer rates, even with fast disks and slow processors. Specifically, they should be faster than 20 Mbytes/sec on a relatively modest modern machine such as the author's Cyrix 6x86MX/266 (which has a clock frequency of 233 MHz). This translates as under 11.7 cycles/byte, within the range of stream ciphers but well outside even the fastest traditional block cipher rates. The current C implementation of Mercy achieves 9 cycles/byte; it is likely that an assembly implementation would do rather better.
- **Memory**: The cipher should refer to as little memory as possible, and certainly less than 4kbytes. In many environments, Mercy's keytables will be stored in unswappable kernel memory; more important however is to minimise the amount of Level 1 cache that will be cleared when the cipher is used. 1536 bytes of storage are used.
- **Simplicity**: Mercy is designed to be simple to implement and to analyse.
- **Decryption**: Decryption will be much more frequent than encryption and should be favoured where there is a choice.

4 Description of Mercy

Since most of Mercy's operations are based around 32-bit words, we define $Z_w = Z_{2^{32}}$. Vectors are indexed from zero, so a vector $P \in Z_w^{128}$ of 128 32-bit numbers will be indexed as $(P_0, P_1, \dots, P_{127})$. The symbol $\oplus$ represents bitwise exclusive OR; where + appears in the figures with a square box around it, it represents addition in the ring Z_w. Least significant and lowest indexed bytes and words appear leftmost and uppermost in the figures.

Note that some details that would be needed to build a specification of Mercy-based file encryption sufficient for interoperability, such as byte ordering within words, are omitted since they are irrelevant for cryptanalytic purposes.

4.1 T Box

The T box ($T : Z_{256} \to Z_w$, Figure 1) is a key-dependent mapping of bytes onto words. N represents multiplicative inverses in $GF(2^8)$ with polynomial base $x^8 + x^4 + x^3 + x + 1$ except that $N(0) = 0$. $d_0 \dots d_7$ are key dependent bijective affine mappings on $GF(2)^8$.

$$T(x) = \sum_{i=0}^{3} 2^{8i} d_{2i+1}[N(d_{2i}[x])]$$

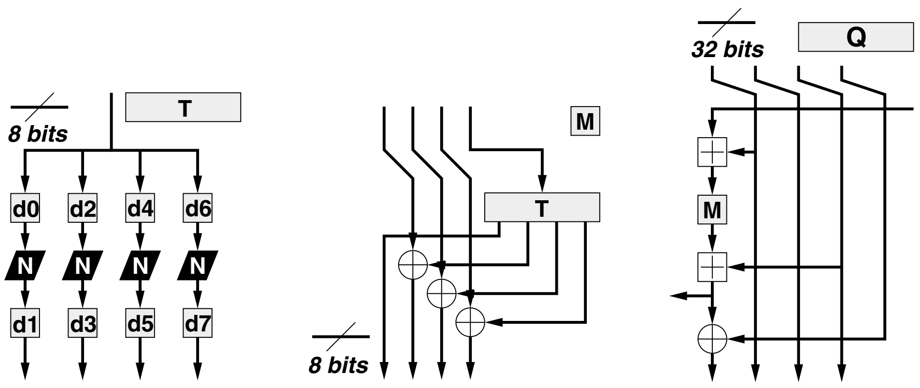


Fig. 1. T box, Operation M, Q state machine

4.2 Operation M

$M : Z_w \to Z_w$ (Figure 1) is drawn from David Wheeler's stream cipher WAKE [21]; it's a simple, key-dependent mapping on 32-bit words. The most significant byte of the input word is looked up in the T box, and the output XORred with the other three bytes shifted up eight bits; the construction of the T box ensures that this mapping is bijective.

$$M(x) = 2^8 x \oplus T\left(\lfloor x/2^{24} \rfloor\right)$$

4.3 Q State Machine

The Q state machine (Figure 1) maps a four-word initial state and one word input onto a four-word final state and one word output ($Q : Z_w^4 \times Z_w \to Z_w^4 \times Z_w$) using taps from a linear feedback shift register and a nonlinear mixing function.

$$Q(S, x) = ((S_3 \oplus y, S_0, S_1, S_2), y) \quad \text{where} \quad y = S_2 + M(S_0 + x) \tag{1}$$

4.4 F Function

The F_n function ($n \geq 8$; Figure 2) accepts a 128-bit spice $G \in Z_w^4$ and a $32n$-bit input $A \in Z_w^n$ and generates a $32n$-bit output $B \in Z_w^n$ ($F_n : Z_w^n \times Z_w^4 \to Z_w^n$). F_{64} (usually just F) is the F function for the Feistel rounds. Here $S_{0 \ldots n+4}$ represents successive 128-bit states of a state machine; $U_{0 \ldots n+3} \in Z_w$ are the successive 32-bit inputs to the state machine, and $V_{0 \ldots n+3} \in Z_w$ are the outputs.

$$\begin{aligned}
F_n(A, G) &= B \quad \text{where} \\
S_0 &= (A_{n-4}, A_{n-3}, A_{n-2}, A_{n-1}) \\
(S_{i+1}, V_i) &= Q(S_i, U_i) \qquad (0 \le i < n + 4) \\
U_i &= \begin{cases} G_i & (0 \le i < 4) \\ A_{i+n-12} & (4 \le i < 8) \\ A_{i-8} & (8 \le i < n) \end{cases} \\
B_i &= \begin{cases} V_{i+8} & (i < n - 4) \\ S_{n+4,i+4-n} & (n - 4 \le i < n) \end{cases}
\end{aligned}$$

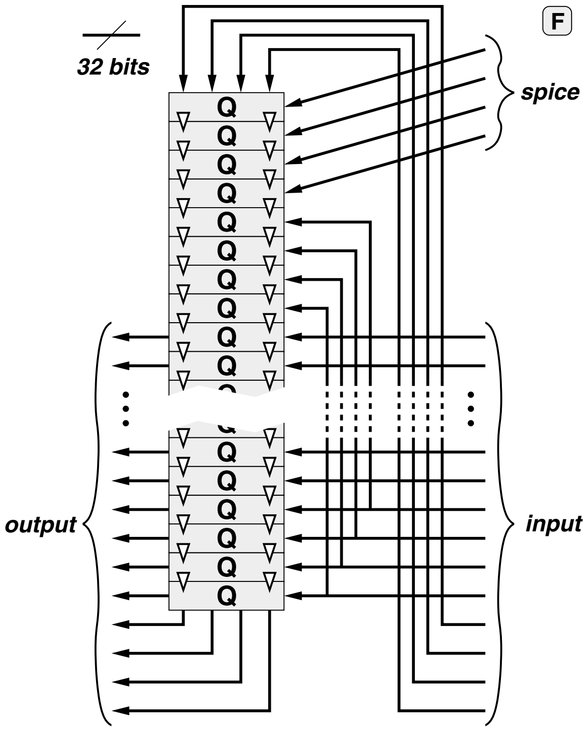


Fig. 2. F function

4.5 Round Structure

Mercy uses a six round Feistel structure (Figure 3) with partial pre- and post-whitening; unusually, the final swap is *not* omitted. The spice $G \in Z_w^4$ (usually the sector number) goes through a "spice scheduling" procedure, analogous with

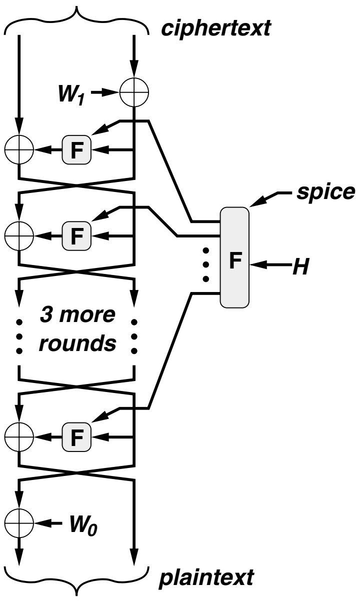


Fig. 3. Round structure (decryption illustrated)

key scheduling, through which the "spice material" $G' \in Z_w^{24}$ is generated based on the input spice, using the F_{24} variant of the F function; this forms six 128-bit "round spices". Spice scheduling uses a dummy input to the F function; for this a vector of incrementing bytes $H \in Z_w^{24}$ is used. $P \in Z_w^{128}$ represents the plaintext, $C \in Z_w^{128}$ the ciphertext, and $W_0, W_1 \in Z_w^{64}$ the whitening values. We describe decryption below; since Mercy is a straightforward Feistel cipher encryption follows in the straightforward way.

$$\begin{aligned}
H_i &= \sum_{j=0}^{3} 2^{8j}(4i+j) \\
G' &= F_{24}(H, G) \\
(L_0, R_0) &= (C_{0\ldots63}, C_{64\ldots127} \oplus W_1) \\
(L_{i+1}, R_{i+1}) &= (R_i, L_i \oplus F_{64}(R_i, G'_{4i\ldots4i+3})) \qquad (0 \le i < 6) \\
(P_{0\ldots63}, P_{64\ldots127}) &= (L_6 \oplus W_0, R_6)
\end{aligned}$$

4.6 Key Schedule

The key is used to seed a CPRNG from which key material is drawn; [10] is used in the sample implementation (after discarding 256 bytes of output), and is convenient since it's small and byte oriented, but any strong CPRNG will serve. Then the procedure in Figure 4 generates the substitutions $d_{0\ldots7}$ and the 2048-bit whitening values W_0, W_1.

```
for $i \leftarrow 0 \ldots 7$ do
    $d_i[0] \leftarrow$ random_byte()
    for $j \leftarrow 0 \ldots 7$ do
        do
            $r \leftarrow$ random_byte()
        while $r \in d_i[0 \ldots 2^j - 1]$
        for $k \leftarrow 0 \ldots 2^j - 1$ do
            $d_i[k + 2^j] \leftarrow d_i[k] \oplus r \oplus d_i[0]$
for $i \leftarrow 0 \ldots 1$ do
    for $j \leftarrow 0 \ldots 64$ do
        $W_{i,j} \leftarrow 0$
        for $k \leftarrow 0 \ldots 3$ do
            $W_{i,j} \leftarrow W_{i,j} + 2^{8k}$random_byte()
```

Fig. 4. Key schedule pseudocode

An expected 10.6 random bytes will be drawn for each d. Once $d_{0\ldots7}$ have been determined, a 1k table representing the T box can be generated. During normal use 1536 bytes of key-dependent tables are used.

5 Design of Mercy

Existing approaches to large block ciphers use a few strong rounds based on existing cryptographic primitives. These ciphers cannot achieve speeds better than half that of the fastest block ciphers [2] or a third of the fastest stream ciphers [1]. Current block cipher speeds don't approach the needs of the design goals even before the extra penalties for doubling up, while those solutions based on stream ciphers pay a heavy penalty in key scheduling overhead that puts their speeds well below those needed.

Mercy addresses this by using more rounds of a weaker function. This makes more efficient use of the work done in earlier rounds to introduce confusion in later rounds, and is closer to a traditional block cipher approach drawn across a larger block. It also draws more state between disparate parts of the block, protecting against the certificational weaknesses identified in Section 2.

5.1 Balanced Feistel Network

Balanced Feistel networks are certainly the best studied frameworks from which to build a block cipher, although I know of no prior work applying them to such large block sizes. They allow the design of the non-linear transformations to disregard efficiency of reversal and provide a familiar framework by which to analyse mixing. Balanced networks seem better suited to large block ciphers than unbalanced networks, since an unbalanced network is likely to have to do work proportional to the larger of the input and output data width.

Feistel ciphers normally omit the final swap, so that decryption has the same structure as encryption. However, Mercy implementations will typically encrypt blocks in-place, and the cost of having an odd number of swaps (forcing a real swap) would be high, so the last swap is not omitted.

Mercy's round function, while weaker than those used in [1], is considerably stronger than that of traditional Feistel ciphers, necessitating many fewer rounds. The larger block size allows more absolute work to be done in each round, while keeping the work per byte small.

5.2 Key Schedule and S-Boxes $d_{0\dots7}$

The function N used in building the T box is that used for nonlinear subsitution in [7]; it is bijective and has known good properties against linear and differential cryptanalysis, such as low LCmax and DCmax in the terminology of [13]. We use this function to build known good key-dependent bijective substitutions using an extension of the technique outlined in [4]; however, rather than a simple XOR, the d mappings before and after N are drawn at random from the entire space of bijective affine functions on $GF(2)^8$, of which there are approximately $2^{70.2}$, by determining first the constant term $d[0]$ and then drawing candidate basis values from the CPRNG to find a linearly independent set. Because the d functions are affine, the LCmax and DCmax of the composite function $d_1 \circ N \circ d_0$ (and siblings) will be the same as those of N itself. The composite functions will also be bijective since each of the components are, and hence all the bytes in each column of the T table will be distinct.

However, there are fewer possible composite functions than there are pairs d_0, d_1. In fact for each possible composite function, there are $255 \times 8 = 2040$ pairs d_0, d_1 which generate it. This follows from the following two properties of N:

$$\forall a \in GF(2^8) - \{0\} : \forall x \in GF(2^8) : aN(ax) = N(x)$$
$$\forall b \in 0 \dots 7 : \forall x \in GF(2^8) : N(x^{2^j}) = N(x)^{2^j}$$

Since both multiplication and squaring in $GF(2^8)$ are linear (and hence affine) in $GF(2)^8$ (squaring because in a field of characteristic 2, $(x+y)^2 = x^2 + xy + yx + y^2 = x^2 + y^2$), both of these properties provide independent ways of mapping from any pair d_0, d_1 to pairs which will generate the same composite function, distinct in every case except $(a, b) = (1, 0)$. Taking this into account, the number of distinct composite functions possible is approximately $2^{129.4}$, and there are $2^{4613.7}$ functionally distinct keys in total (considering W_0, W_1 as well as T).

Little attention has been paid to either the time or memory requirements of the key schedule, since key scheduling will be very infrequent and typically carried out in user space.

5.3 Operation M

As with [21], this operation is bijective, since from the least significant byte of the output the input to the T box can be inferred. We differ from [21] in using the most significant byte for the lookup rather than the least; this slightly improves the mixing gained from addition and seems to be very slightly faster on the author's machine.

5.4 Q State Machine

A key-dependent state machine is an efficient way to bring about strong dependencies between widely separated parts of the block; since the state machine is reversible, changing a given input word guarantees that every subsequent state will be different and makes it slightly more likely than chance that every output word will be different.

The basis of the state machine is 32 parallel four-stage LFSRs based on the polynomial x^4+x^3+1. The input to each LFSR is provided by a nonlinear mixing chain based on carry bits from other LFSRs and taps into the state which are then fed into Operation M after addition-based mixing with the input. The use of an LFSR ensures that any pair of distinct inputs of the same length which leave the LFSR in the same state must be at least 5 words long.

Every T box lookup depends on the previous lookup, even across rounds. This goes against the design principles outlined in [18,5] which suggest that ciphers should be designed to make best use of the parallelism that modern processors can achieve, and to be wary of the memory latency of table lookups. A variant on Q which allows several T box lookups to take place in parallel by taking taps from later in the LFSR is easy to construct, but surprisingly did not result in any speed improvements on the target platform. Ciphers similar to Mercy which use this technique to improve parallelism may be appropriate for other architectures.

Since the Feistel rounds use XOR mixing, Q is also designed such that the first operation on the input is addition, as is the last operation on the output. This improves operation mixing, helping to frustrate attacks which model the cipher using a single algebraic group. XOR is also used within Q for the same reason.

The output is chosen for the property that, where $Q(S,x) = (S',y)$, if either of S or S' is known and either of x or y is known, the two unknowns can be inferred. We prove this in four cases below:

1. S, x known, S', y unknown: use Q directly.
2. S', x known, S, y unknown: from Equation 1 we infer $y = S'_3 + M(S'_1 + x)$ and $S = (S'_1, S'_2, S'_3, S'_0 \oplus y)$.
3. S, y known, S', x unknown: $x = M^{-1}(y - S_2) - S_0$ (defined since M is bijective), then apply Q as normal.
4. S', y known, S, x unknown: find S as in 2 and x as in 3.

5.5 F Function

These properties of Q are used to build a fast, bijective F function. A bijective F function lends resistance to a number of attacks, making 2 round iterative differential characteristics impossible and frustrating truncated differential cryptanalysis, as well as lending confidence that the input is being used efficiently. For a fixed spice G, we infer F's input A given the output B as follows: the final state of the state machine is simply the last four words of the output: $S_{n+4} = B_{n-4...n-1}$; from this, we can run the state machine backwards up to S_8 as with point 4 of Section 5.4 above, inferring $A_{0...n-5}$ as we do so. We then use $A_{n-8...n-5}$ (which we just inferred) and G to infer S_0 using point 2 of Section 5.4 above, which gives us our last four inputs $A_{n-4...n-1}$.

5.6 Avalanche

This F function does not provide full avalanche between input and output. I see no secure way to build a balanced full-avalanche F function for large blocks that isn't so much slower than Mercy's F function that the time cost would not be better spent on more rounds with the weaker F function.

Instead of providing full avalanche, the F function makes two weaker guarantees that together are almost as effective:

– A change to any bit of the input will on average change half the bits of the last 128 bits of the output
– A change to any bit of the last 128 bits of the input will on average change half the bits of the output

A full avalanche F function achieves avalanche across the whole block after three rounds. This construction does so after four rounds. In addition, in encryption and decryption every keytable lookup depends on the result from the previous lookup.

The partial collision attack from Section 2 will demonstrate that after six rounds Mercy's avalanche is imperfect, since only 384 bits of state are carried between some parts of the block, but such an attack would require that 2^{193} chosen plaintexts be encrypted, and is thus outside the security goals of the cipher. This suggests a distinction between perfect avalanche, and avalanche sufficient to defeat cryptanalysis; this distinction is useful since mechanisms for providing perfect avalanche, such as networks based on Fourier transforms (used in SAFER [12] and proposed by [16] for large block ciphers), can carry a heavy performance penalty on large blocks. This distinction is not useful on small blocks: if this attack is possible against a cipher with a 64-bit block, it will not require more than 2^{33} chosen plaintexts.

5.7 Whitening

Mercy only whitens one half of the input and output, since the cost both in time and storage of whitening both halves would be significant, and since the primary function of whitening is to hide the input of the F function from attackers. On large block ciphers, whitening also serves to frustrate attacks based on creating inputs with special structures, such as attempts to induce a repeating state in the state machine of F.

5.8 Linear and Differential Cryptanalysis

We do not prove Mercy resistant to either linear or differential cryptanalysis; a large block cipher meeting the performance goals that could be proven resistant would be a worthy goal. However, four features of the cipher lend resistance to these classes of attack.

First, the key dependent subsitutions are optimised against linear and differential cryptanalysis as described in Section 5.2.

Second, the LFSR-based construction of the Q state machine forces any input to the F function with active substitutions (in the terminology of [7]) to make at least three substitutions active. In practice, the intent of the F function design is that any difference in the input causing a difference in a T box substitution will cause all subsequent T box subsitution to be uncorrelated; avoiding this effect will be very hard for attackers. Most F functions cannot afford the luxury of 68 chained non-linear substitutions.

Third, the initial and final whitening should force attackers to look for difference propagations or linear correlations over at least four rounds.

Fourth, the ways in which key material is introduced in the F function should mean that inferring a suggested key from a given plaintext-ciphertext pair should be extremely difficult.

6 Conclusions

Large blocks are useful in any application where random access to data is desirable, of which sector encryption is a prime example. Mercy is intended to demonstrate the possibility of building an efficient block cipher for large blocks. Mercy's design was inspired by two beliefs:

- Large block sizes can lend useful advantages in security and speed
- Avalanche across large blocks need not be perfect before a cryptanalyst with limited computing resources cannot distinguish it from perfect, as explained in Section 5.6.

However, the primary motivation for the design of Mercy is not that the cipher be directly used for this application; it is to inspire further work where it is badly needed. Disk encryption suffers none of the barriers to adoption from interoperability suffered by (for example) email encryption. But it is very rarely used, and the main barrier to adoption among security conscious users is the high performance penalties it currently exacts. I hope that Mercy demonstrates that fast, secure large block ciphers are possible, and inspires better cryptographers to propose better designs.

Acknowledgements. Thanks are due to the anonymous reviewers both on this submission and on submission to Crypto ’99, who provided valuable references and suggestions, to Stefan Lucks for his sound advice, and to David Wagner, David A Molnar, Conrad Hughes, and Keith Lockstone for their comments on an earlier design and for encouragement. Special thanks to Scott Fluhrer who has found a very effective differential attack on the cipher presented here, details of which are on the Mercy home page.

References

1. Ross Anderson and Eli Biham. Two practical and provably secure block ciphers: BEAR and LION. In Gollman [8], pages 113–120.
2. Mihir Bellare and Phillip Rogaway. On the construction of variable-input-length ciphers. In Lars R. Knudsen, editor, *Fast Software Encryption: 6th International Workshop*, volume 1636 of *Lecture Notes in Computer Science*, pages 231–244, Rome, Italy, March 1999. Springer-Verlag.
3. Eli Biham, editor. *Fast Software Encryption: 4th International Workshop*, volume 1267 of *Lecture Notes in Computer Science*, Haifa, Israel, 20–22 January 1997. Springer-Verlag.
4. Eli Biham and Alex Biryukov. How to strengthen DES using existing hardware. In Josef Pieprzyk and Reihanah Safavi-Naini, editors, *Advances in Cryptology—ASIACRYPT ’94*, volume 917 of *Lecture Notes in Computer Science*, pages 398–412, Wollongong, Australia, 28 November–1 December 1994. Springer-Verlag.
5. Craig S. K. Clapp. Optimizing a fast stream cipher for VLIW, SIMD, and superscalar processors. In Biham [3], pages 273–287.
6. Joan Daemen and Craig S. K. Clapp. Fast hashing and stream encryption with Panama. In Serge Vaudenay, editor, *Fast Software Encryption: 5th International Workshop*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74, Paris, France, 23–25 March 1998. Springer-Verlag.
7. Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. NIST AES Proposal, 1998.
8. Dieter Gollman, editor. *Fast Software Encryption: Third International Workshop*, volume 1039 of *Lecture Notes in Computer Science*, Cambridge, UK, 21–23 February 1996. Springer-Verlag.
9. Peter Gutmann. Secure filesystem. `http://www.cs.auckland.ac.nz/%7Epgut001/sfs/index.html`, 1996.
10. K. Kaukonen and R. Thayer. A stream cipher encryption algorithm “ARCFOUR”. Internet-Draft draft-kaukonen-cipher-arcfour-03.txt, July 1999. The draft is a work in progress, but the algorithm (as RC4(tm)) is due to Ronald L. Rivest.
11. Stefan Lucks. BEAST: A fast block cipher for arbitrary blocksizes. In *FIP TC-6 and TC-11 Joint Working Conference on Communications and Multimedia Security*, September 1996.
12. James L. Massey. SAFER K-64: A byte-oriented block-ciphering algorithm. In Preneel [15]. Published 1995.
13. Mitsuru Matsui. New structure of block ciphers with provable security against differential and linear cryptanalysis. In Gollman [8], pages 205–218.
14. Chris J. Mitchell. Authenticating multicast Internet electronic mail messages using a bidirectional MAC is insecure. In *IEEE Transactions on Computers*, number 41, pages 505–507. 1992.

15. Bart Preneel, editor. *Fast Software Encryption: Second International Workshop*, volume 1008 of *Lecture Notes in Computer Science*, Leuven, Belgium, 14–16 December 1994. Springer-Verlag. Published 1995.
16. Terry Ritter. A mixing core for block cipher cryptography. `http://www.io.com/%7Eritter/MIXCORE.HTM`, 1998.
17. Phillip Rogaway and Don Coppersmith. A software-optimized encryption algorithm. In Ross Anderson, editor, *Fast Software Encryption*, pages 56–63. Springer-Verlag, 1994.
18. Bruce Schneier and Doug Whiting. Fast software encryption: Designing encryption algorithms for optimal software speed on the Intel Pentium processor. In Biham [3], pages 242–259.
19. Rich Schroeppel. Hasty Pudding Cipher specification. NIST AES Proposal, June 1998.
20. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.
21. David Wheeler. A bulk data encryption algorithm. In Preneel [15]. Published 1995.

A Statistical Attack on RC6

Henri Gilbert[1], Helena Handschuh[2], Antoine Joux[3], and Serge Vaudenay[4]

[1] France Telecom
[2] Gemplus
[3] SCSSI
[4] Ecole Normale Supérieure – CNRS
Contact Helena.Handschuh@gemplus.com

Abstract. This paper details the attack on RC6 which was announced in a report published in the proceedings of the second AES candidate conference (March 1999). Based on an observation on the RC6 statistics, we show how to distinguish RC6 from a random permutation and to recover the secret extended key for a fair number of rounds.

1 Introduction

RC6 is one of the 15 candidate algorithms that were presented at the first Advanced Encryption Standard candidate conference in August 1998. It was submitted by RSA laboratories [9] and has been selected as one of the five finalists for the second round of the AES contest organized by NIST [1].

In this paper, we first show the existence of a statistical weakness in RC6 which allows to mount a distinguisher attack on a reduced number of rounds. This means that given a certain number of plaintext-ciphertext pairs, an attacker is able to distinguish RC6 from a random permutation. A distinguisher for the r-round version of a cipher may often be converted into a key-recovery attack on $r+1$ or even more rounds. Matsui's linear cryptanalysis of DES provides a typical example of such a situation [7]. This also holds for RC6 : we show that we can gain one round as compared with our distinguisher to recover the extended keys of RC6 reduced to 14 rounds (or equivalently 15 RC6 inner rounds).

The paper is organised as follows : in the next Section we give the outlines of RC6 and in Section 3 we present the probabilistic event which leaks information. In Section 4 we explicitely construct the distinguisher and in Section 5 we adapt the latter to recover the extended secret key. Finally, we shortly discuss the case of RC5 and conclude.

2 RC6 Outlines

RC6 is characterized by three parameters (w, r, b). It is dedicated to w-bit microprocessors and encrypts $4w$-bit blocks by using four registers. (We assume that w

is an integral power of 2.) It has r rounds and uses a b-byte secret key. The nominal parameters for AES are $(32, 20, 16)$, $(32, 20, 24)$ and $(32, 20, 32)$, respectively for a 128, 196 and 256-bit user key. There is a key scheduling algorithm which extends the original b-byte key into an $2r + 4$-word array $S = (S_0, \ldots, S_{2r+3})$. In this paper, we will only use the w and r parameters, so we consider that the encryption is performed by using an arbitrary $2r + 4$-word array S which plays the role of the secret key.

The encryption is performed by using four registers A, B, C, D. The algorithm is described by the following pseudo-code.

```
Input: $(A, B, C, D)$
    1. $B \leftarrow B + S_0$, $D \leftarrow D + S_1$
    2. for $i = 1$ to $r$ do
            $A \leftarrow ((A \oplus f(B)) \lll f(D)) + S_{2i}$
            $C \leftarrow ((C \oplus f(D)) \lll f(B)) + S_{2i+1}$
            $(A, B, C, D) \leftarrow (B, C, D, A)$
    3. $A \leftarrow A + S_{2r+2}$, $C \leftarrow C + S_{2r+3}$
Output: $(A, B, C, D)$
```

Here the f function plays the role of a pseudo-random generator defined by

$$f(x) = g(x) \bmod 2^w \lll log_2 w = x(2x + 1) \bmod 2^w \lll log_2 w \ .$$

A picture of the RC6 encryption algorithm is given hereafter.

RC6 is very similar to RC5 in that it uses only simple operations such as binary addition, exclusive or and circular rotations. In addition, RC6 performs a simple modular multiplication.

Our results show that a reduced number of rounds of RC6 may be distinguished from a random permutation, which in turn enables an attacker to recover the secret keys of RC6 with one more round. This analysis also partly transposes to RC5. We would like to mention that an outline of our attack was introduced for the first time at the second AES conference in Rome in March 1999 [2], and that another paper dealing with the same kind of RC6 statistics [6] appears in these proceedings. However, the work reported in [6] and the work reported here are quite independent, both approaches for handling the RC6 statistics differ to some extent, and we feel it is important to present the attack announced in [2] in details here. Interestingly, both papers show that we can distinguish RC6 from a random permutation in polynomial time for a fair number of rounds, although it has been made clear [4,8] that the RC6 frame provides a pseudorandom permutation after five rounds once the data-dependent rotations are removed.

3 A Probabilistic Event on RC6 Encryption

For $1 \leq i \leq r$ and $0 \leq j < 4$, we let $R_{i,j}(S, a, b, c, d)$ denote the value of the register with index j (considering that index 0 is for A, index 1 is for B, ...) after

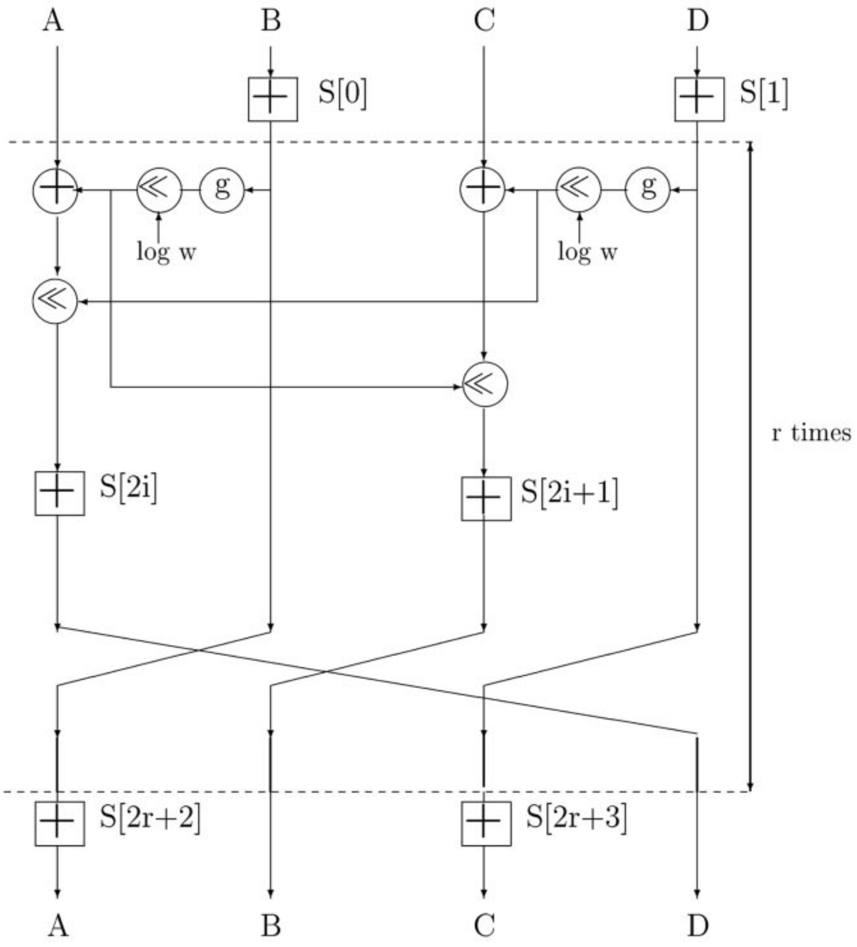


Fig. 1. Encryption with RC6-$w/r/b$ where $g(x) = x \times (2x + 1)$.

the ith round when the input of the encryption is (a, b, c, d) and the key is S. We can extend the above notation to $i = 0$ letting $R_{0,j}(S, a, b, c, d)$ denote the input words to the first round. We will omit (S, a, b, c, d) in most cases. In the sequel we implicitely assume that the j index is taken modulo 4. We start with the following simple fact.

Lemma 1. *For any i and any (S, a, b, c, d), we have*

$$\left.\begin{array}{l} f(R_{i-1,1}) \equiv 0 \pmod{w} \\ f(R_{i-1,3}) \equiv 0 \pmod{w} \end{array}\right\} \Longrightarrow \left\{\begin{array}{l} R_{i,3} - R_{i-1,0} \equiv S_{2i} \pmod{w} \\ R_{i,1} - R_{i-1,2} \equiv S_{2i+1} \pmod{w} \end{array}\right.$$

and in addition, $R_{i,0} = R_{i-1,1}$ and $R_{i,2} = R_{i-1,3}$.

This comes from the fact that if the mod w part of $f(B)$ and $f(D)$ are both zero in the ith round, then nothing is XORed onto the mod w part of A and C, and none are rotated.

This fact extends into the following

Lemma 2. *If we have $f(R_{i-1,1}) \equiv f(R_{i-1,3}) \equiv 0 \pmod{w}$ for $i = k, k+2, \ldots, k+2\ell$, then $R_{k+2\ell,-2\ell-1} - R_{k-1,0} \bmod w$ and $R_{k+2\ell,1-2\ell} - R_{k-1,2} \bmod w$ are constants which only depend on S.*

Assuming that the outputs of f mod w behave like random numbers, this event holds with probability $w^{-2\ell}$. We thus have the following heuristic result which has been confirmed by statistical experiments for $w = 32$ and small values of r.

Theorem 1. *Under heuristic assumptions, there exists some functions $c_1(S)$ and $c_2(S)$ such that for random $(R_{0,0}, \ldots, R_{0,3})$ and a random S we have*

$$\Pr\begin{bmatrix} R_{r,1-r}(S) - R_{0,1}(S) \bmod w = c_1(S) \\ R_{r,3-r}(S) - R_{0,3}(S) \bmod w = c_2(S) \end{bmatrix} \approx w^{-2}\left(1 + w^{-2\lfloor\frac{r}{2}\rfloor}\right).$$

4 On Distinguishing RC6 from a Random Permutation

We can construct a distinguisher between RC6 and a random permutation by using the above theorem through a known plaintext attack.

1. The distinguisher first gets n random samples $(x_i, \mathrm{Enc}(x_i))$ where

$$x_i = (x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3})$$

   and

$$\mathrm{Enc}_i = (y_{i,0}, y_{i,1}, y_{i,2}, y_{i,3}).$$

2. Then it hashes the samples onto

$$h_i = (y_{i,1-r} - x_{i,1} \bmod w, y_{i,3-r} - x_{i,3} \bmod w).$$

3. It then creates w^2 counters which correspond to possible h_i values and counts the number $n_{(u,v)}$ of i indices such that $h_i = (u, v)$.
4. If the maximum of all $n_{(u,v)}$ is greater than a given threshold t, output 1, otherwise, output 0.

We let $\epsilon \approx w^{-2\lfloor\frac{r}{2}\rfloor}$ denote the probability that the event of Theorem 1 occurs for RC6. We need to compute the advantage in terms of n, t, ϵ of this attack for distinguishing RC6 from a random permutation.

Let us choose $t = n.w^{-2} + \delta$. (nw^{-2} is the expected value of one counter for random hashes so δ measures the deviation from the ideal expected case.)

The probability p that the distinguisher outputs 1 for RC6 is greater than the probability that the counter which corresponds to the constant values in

Theorem 1 is greater than t. When n is large, this counter tends towards a normal law with expected value $n(w^{-2}(1-\epsilon)+\epsilon)$ (which we approximate by $nw^{-2}+n\epsilon$) and variance approximately nw^{-2}. We let

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt.$$

We have

$$p \approx \varphi\left(-\frac{t - nw^{-2} - n\epsilon}{\sqrt{n}w^{-1}}\right) + \left(1 - \varphi\left(\frac{t - nw^{-2}}{\sqrt{n}w^{-1}}\right)^{w^2-1}\right)$$

which is

$$p \geq \varphi\left(-\frac{t - nw^{-2} - n\epsilon}{\sqrt{n}w^{-1}}\right).$$

This means

$$p \geq \varphi\left(-\delta w n^{-\frac{1}{2}} + w\epsilon\sqrt{n}\right). \quad (1)$$

Now the probability p^* that the distinguisher outputs 1 for a random permutation is less that w^2 times the probability that one given counter is greater then t. This counter tends to behave like a normal law with expected value nw^{-2} and variance nw^{-2}. We thus have

$$p^* \leq w^2 \varphi\left(-\frac{t - nw^{-2}}{\sqrt{n}w^{-1}}\right)$$

which means

$$p^* \leq w^2 \varphi\left(-\delta w n^{-\frac{1}{2}}\right). \quad (2)$$

Therefore the advantage for distinguishing RC6 from a random permutation is

$$\mathrm{Adv} \geq \varphi\left(-\delta w n^{-\frac{1}{2}} + w\epsilon\sqrt{n}\right) - w^2 \varphi\left(-\delta w n^{-\frac{1}{2}}\right).$$

If we derive this function with respect to δ, we obtain the maximum when the derivative is equal to zero. The choice of δ which maximizes this right hand term is

$$\delta = \frac{2\log w}{\epsilon w^2} + \frac{\epsilon n}{2}$$

for which

$$\mathrm{Adv} \geq \varphi\left(-\frac{2\log w}{\epsilon w\sqrt{n}} + \frac{\epsilon w\sqrt{n}}{2}\right) - w^2 \varphi\left(-\frac{2\log w}{\epsilon w\sqrt{n}} - \frac{\epsilon w\sqrt{n}}{2}\right).$$

This analysis leads to the following result.

Theorem 2. *Let $\alpha = \frac{\epsilon w \sqrt{n}}{2\sqrt{\log w}}$. Under heuristic assumptions, the above distinguisher, when used with*

$$t = \frac{n}{w^2} + \frac{2\log w}{\epsilon w^2} + \frac{\epsilon n}{2} \quad \text{and} \quad n \geq 4\alpha^2 w^{4\lfloor \frac{r}{2} \rfloor - 2} \log w$$

has an advantage greater than

$$\mathrm{Adv} \geq \varphi\left(\sqrt{\log w}(-\alpha^{-1} + \alpha)\right) - w^2 \varphi\left(\sqrt{\log w}(-\alpha^{-1} - \alpha)\right).$$

Considering $\alpha = 5$ we have

$$\mathrm{Adv} \geq \varphi\left(\frac{24}{5}\sqrt{\log w}\right) - w^2 \varphi\left(-\frac{26}{5}\sqrt{\log w}\right)$$

with a complexity of

$$n \geq 100 w^{4\lfloor \frac{r}{2} \rfloor - 2} \log w.$$

We have to be concerned that the total number of samples cannot be greater than 2^{4w}, which is the total number of possible plaintexts. Hence the above attack is significant for

$$r \leq 2\left\lfloor \frac{4w - 7 - \log_2 \log w}{4 \log_2 w} + \frac{1}{2} \right\rfloor + 1 \ .$$

As an application, with the nominal choice $w = 32$ we obtain an advantage greater than $1 - 2^{-60}$ with a complexity of $n \approx 2^{20\lfloor \frac{r}{2} \rfloor - 2}$. Thus we can break up to $r = 13$ rounds (with $n = 2^{118}$).

5 On Recovering the Secret Key

5.1 A Simplified Approach for Recovering S_0 and S_1

Let us focus on nominal RC6 reduced to $r = 14$ rounds for a moment. Then a way of adapting the distinguisher to recover the whole secret key for RC6 reduced to 14 rounds is by a known plaintext attack which proceeds in the following way.

Suppose we black box encrypt a multiple m of the n plaintexts required by the previously described distinguisher on 13 rounds, thus obtaining m (x_i, y_i) plaintext-ciphertext pairs for the 14-round RC6. Let $\Delta A = A_{\text{out}} - A_{\text{in}} \pmod{w}$ where $A_{\text{out}} = y_{i,-14}$ and $A_{\text{in}} = x_{i,0}$ denote the input-output difference of the $\log_2 w$ least significant bits of the input word A and similarly let $\Delta C = C_{\text{out}} - C_{\text{in}} \pmod{w}$ where $C_{\text{out}} = y_{i,2-14}$ and $C_{\text{in}} = x_{i,2}$ denote the difference modulo w on input word C. For those (x_i, y_i) pairs such that the A and

C input words are not rotated at the first round, ΔA and ΔC are equal, up to the unknown constants $S_2 \pmod w$ and $S_3 \pmod w$, to the h_i differences considered in the 13-rounds distinguisher of Section 4.

The exhaustive trial of all the S_0 and S_1 keys, i.e. the computation for each (S_0, S_1) key assumption, of the $(\Delta A, \Delta C)$ frequencies distribution on the subset of plaintext-ciphertext pairs such that no A and C rotations occur at the first round, followed by the 13-round distinguisher test of Section 4, can be performed in an efficient way which avoids processing each plaintext individually for each key assumption.

• First we generate a table of $2^{2w+2\log_2 w}$ (e.g. 2^{74} for nominal RC6) elements, where each entry is the frequency observed for the plaintext-ciphertext pairs according to the value of the B and D input words as well as the ΔA and ΔC input-output differences modulo w (i.e. a potential value of the constant differences if all the rotations were zero as in our model).

• Now for approximately $2^{w-\log_2 w}$ (e.g. 2^{27}) "good" B values, we obtain that $f(B+S_0) \equiv 0 \pmod w$ in the first round. Therefore for each possible choice of the first subkey S_0, we may add together the frequencies of the $2^{w-\log_2 w}$ corresponding good B values. This requires a work load of about $2^{w-\log_2 w+w+2\log_2 w} = 2^{2w+\log_2 w}$ (e.g. 2^{69}) operations per S_0 guess. We are left with a table of $2^{w+2\log_2 w}$ - e.g. 2^{42} - $(D, \Delta A, \Delta C)$ frequencies.

• Next, for every possible S_1 value, we can do the same. For a given guess, we select the $2^{w-\log_2 w}$ possible values for D which achieve $f(D+S_1) \equiv 0 \pmod w$ in the first round, and add their frequencies together. We are left with a table of w^2 $(\Delta A, \Delta C)$ frequencies, the maximum of which corresponds to the sum of some key bits when the two subkeys are correctly guessed. This step requires an effort of $2^{w+\log_2 w}$ operations for all (S_0, S_1) subkey guesses.

• Once such a table of frequencies of the $(\Delta A, \Delta C)$ values has been obtained, the distinguisher of Section 4 may be applied quite naturally to it. If (S_0, S_1) is the correct subkey guess, one of the frequencies is expected to pass the test, whereas the test is expected to fail when wrong values have been picked. Thus this procedure allows us to recover the first two subkeys using a memory of less than $2 \cdot 2^{2w+2\log_2 w}$ words (e.g. 2^{75}) and a workload

$$C = 2^w \left(2 \cdot 2^{2w+\log_2 w}\right) = 2^{3w+\log_2 w+1} \quad ,$$

(e.g. 2^{102}), which is far less than the number of encryptions needed for the distinguisher anyway. However, using this technique, we filter out about w^2

plaintext-ciphertext pairs, therefore we have to start off with far more pairs at the beginning of the attack in order to make sure the distinguisher gets enough information after the filtering phase. This leads to a required number of known plaintexts :

$$m \geq 100w^{4\lfloor\frac{r-1}{2}\rfloor} \log w.$$

Note that for $w = 32$ and $r = 14$, m is about equal to the 2^{128} limit.

5.2 Improved Approach Without Filtering

As we saw in the last section the fact that we filter pairs for which the first two rotations are zero "costs" a factor w^2 in the number of plaintext-ciphertext pairs. We want to avoid this and use only the n pairs required by the distinguisher. We actually guess the first two rotations at the cost of some more memory.

• Let $\beta = f(B) \pmod w$ and $\delta = f(D) \pmod w$ be the two rotations of the first round. For each of the w^2 potential values of (β, δ), we generate a hash table for the frequencies of the tuples

$$\big(B, D, (A_{\text{in}} \lll \delta) \bmod w, A_{\text{out}} \bmod w, (C_{\text{in}} \lll \beta) \bmod w, C_{\text{out}} \bmod w\big)$$

Thus we have w^2 tables of size $2^{2w+4\log_2 w}$ (e.g. 2^{84}) each, giving all the frequencies for the various potential (β, δ) couples of rotations. Note that the generation of such tables may be optimized (avoiding an extra work factor of w^2 for each plaintext-ciphertext pair) in a way which will be discussed below.

• Now for every guess of S_0, for each of the w possible β values, we may select the $2^{w-\log_2 w}$ (e.g. 2^{27}) B values such that $f(B+S_0) = \beta \bmod w$ and, for each of the w potential δ values, add together, in the (β, δ) table, the frequencies of those t-uples for which the values of D, $\Delta A = (A_{\text{out}} - ((A_{\text{in}} \oplus f(B+S_0)) \lll \delta)) \bmod w$, $C_{\text{in}} \lll \beta \bmod w$ and $C_{\text{out}} \bmod w$ are the same. We thus obtain w^2 tables providing $(D, \Delta A, C_{\text{in}} \lll \beta \bmod w, C_{\text{out}} \bmod w)$ frequencies, at the expense of a $2^{2w+5\log_2 w}$ (e.g. 2^{89}) work load per S_0 assumption.

• Next, for each guess of S_1, for each of the w possible δ values, we may select the $2^{w-\log_2 w}$ (e.g. 2^{27}) D values such that $f(D+S_1) = \delta \bmod w$ and, for each of the w potential β values, add together, in the (β, δ) table derived at the former step, the frequencies of those $(D, \Delta A, C_{\text{in}} \lll \beta \bmod w, C_{\text{out}} \bmod w)$ tuples for which the values of ΔA and $\Delta C = (C_{\text{out}} - ((C_{\text{in}} \oplus f(D+S_1)) \lll \beta)) \bmod w$ are the same. By adding up all the $(\Delta A, \Delta C)$ frequencies obtained for all the (β, δ) pairs, we are left with a table of w^2 $(\Delta A, \Delta C)$ frequencies which can be used as an input to the distinguisher of Section 4. The distinguisher is expected to succeed only when the two subkeys S_0 and S_1 are correctly guessed. Thus the above procedure provides the first two subkeys. The work load for this step is about $2^{w+4\log_2 w}$ for each (S_0,S_1) assumption.

Discussion. In order to optimize the generation of the w^2 (β,δ) tables of the (B, D, $A_{\text{in}} \lll \delta \bmod w$, $A_{\text{out}} \bmod w$, $C_{\text{in}} \lll \beta \bmod w$, $C_{\text{out}} \bmod w$) frequencies, we suggest the following technique. We denote by A_L and C_L (resp A_H and C_H) the $w/2+\lfloor\frac{\log_2 w}{2}\rfloor$ (e.g. 18) lowest (resp highest) weight bits of A_{in} and C_{in}. From the n plaintext-ciphertext pairs used in the attack, we first derive the four tables containing the (B, D, A_L, $A_{\text{out}} \bmod w$, C_L, $C_{\text{out}} \bmod w$), (B, D, A_L, $A_{\text{out}} \bmod w$, C_H, $C_{\text{out}} \bmod w$), (B, D, A_H, $A_{\text{out}} \bmod w$, C_L, $C_{\text{out}} \bmod w$), and (B, D, A_H, $A_{\text{out}} \bmod w$, C_H, $C_{\text{out}} \bmod w$) frequencies. Each of the w^2 (β,δ) tables of (B, D, $A_{\text{in}} \lll \delta \bmod w$, $A_{\text{out}} \bmod w$, $C_{\text{in}} \lll \beta \bmod w$, $C_{\text{out}} \bmod w$) frequencies can then be deduced from one of the four above tables. This way, we process the n samples only once, and the additional complexity factor of w^2 corresponding to all possible choices for (β,δ) will apply essentially to the number of entries in each table, which is about $2^{3w+2\lfloor\frac{\log_2 w}{2}\rfloor+2\log_2 w}$ For example, for $w = 32$ and $n = 2^{118}$, this complexity is about $2^{10} \cdot 2^{110}$ instead of $2^{10} \cdot 2^{118}$.

The complexity of the entire procedure for recovering the first two subkeys S_0 and S_1 is less than $2 \cdot 2^{3w+5\log_2 w}$ (e.g. 2^{122}).

Once the first two subkeys are found, we can decrypt one round using the data in the previously described tables and may apply the same technique on the next two subkeys. As we go on recovering the extended key piece by piece, the required number of plaintext-ciphertext pairs to make the distinguisher work decreases very fast. Thus the overall complexity of this attack stays well below the effort of an exhaustive search for the key.

6 On the Existence of Similar RC5 Statistics

RC6 is an enhancement of the RC5 encryption algorithm. RC5 is characterized by three parameters w (word size ; note that the RC5 block size is $2w$), r (number of rounds ; unlike an RC6 round, an RC5 round consists of two half rounds) and b (number of key bytes).

The following statistical property of RC5 is closely related to the RC6 properties summarised in Section 3 above : if, in ρ consecutive RC5 half rounds, the rotation amounts applied at each second half round are all equal to zero, then after ρ half rounds the $\log_2 w$ lowest weight bits of one of the two plaintext halfes A and

B has been simply added (modulo w) with a constant value derived from the key.

The analysis of Section 4 is easy to transpose, to show that ρ half rounds of RC5 can be distinguished from a random permutation using a number n of known plaintexts which stays within a small factor of $w^{2\lfloor\frac{\rho}{2}\rfloor-1}$. For sufficiently low r values, this distinguisher can be used to guess the RC5-$w/r/b$ expanded key, using a number n of known plaintexts which stays within a small factor of $w^{2(r-1)-1}$. However, for usual RC5 parameter choices such as $r = 12$ and a 64-bit block size, the number of available plaintexts is far too low to mount such an attack.

There are some connections beween the above outlined RC5 attack and the RC5 linear attacks mentioned in [5], which require about $4w^{2(r-1)}$ known plaintexts. Both approaches are based related RC5 properties, and the main difference consists in handling $log_2 w$-bit statistics versus binary statistics. We conjecture - but are not fully sure, since we did not check the RC5 key derivation details - that the treatment of $log_2 w$-bit statistics might provide a slight performance improvement over the linear cryptanalysis approach.

7 Conclusion

Extending the work presented at the second AES conference, we have shown the existence of a special statistical phenomenon on RC6 which enables to mount a distinguisher attack on up to 13 rounds. As usual, this kind of attack is shown to be convertible into a known plaintext attack which can break up to 14 rounds of RC6 (or equivalently 15 inner rounds with or without post-whitening), requiring about 2^{118} known plaintexts, 2^{112} memory and a work load of 2^{122} operations. Of course this attack is not anywhere near practical, but still leads us to the conclusion that due to the existence of a slight but iterative statistical weakness in its round function, RC6 does not have a very conservative number of rounds.

References

1. http://www.nist.gov/aes
2. O. Baudron, H. Gilbert, L. Granboulan, H. Handschuh, A. Joux, P. Nguyen, F. Noilhan, D. Pointcheval, T. Pornin, G. Poupard, J. Stern, S. Vaudenay, "Report on the AES Candidates," *The Second Advanced Encryption Standard Candidate Conference,* N.I.S.T., 1999, pp. 53–67.
3. FIPS 46, *Data Encryption Standard,* US Department of Commerce, National Bureau of Standards, 1977 (revised as FIPS 46–1:1988; FIPS 46–2:1993).
4. T. Iwata, K. Kurosawa, "On the Pseudorandomness of AES Finalists – RC6 and Serpent", These proceedings.

5. B. S. Kaliski Jr., Y. L. Yin, “On the Security of the RC5 Encryption Algorithm”, RSA Laboratories Technical Report TR-602, Version 1.0 - September 1998.
6. L. Knudsen, W. Meier, “Correlations in RC6 with a reduced number of rounds ”, These proceedings.
7. M. Matsui, “The first experimental cryptanalysis of the Data Encryption Standard”. In *Advances in Cryptology - Crypto'94*, pp 1-11, Springer Verlag, New York, 1994.
8. S. Moriai, S. Vaudenay, “Comparison of randomness provided by several schemes for block ciphers”, Preprint, 1999.
9. R.L. Rivest, M.J.B. Robshaw, R. Sidney and Y.L. Yin,“The RC6 Block Cipher”, v1.1, August 20, 1998.
10. S. Vaudenay, “An experiment on DES - Statistical Cryptanalysis”. In *3rd ACM Conference on Computer Security*, New Dehli, India, pp139-147, ACM Press, 1996.

Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent

John Kelsey[1], Tadayoshi Kohno[2⋆], and Bruce Schneier[1]

[1] Counterpane Internet Security, Inc.
{kelsey,schneier}@counterpane.com
[2] Reliable Software Technologies
kohno@rstcorp.com

Abstract. We introduce a new cryptanalytic technique based on Wagner's boomerang and inside-out attacks. We first describe this new attack in terms of the original boomerang attack, and then demonstrate its use on reduced-round variants of the MARS core and Serpent. Our attack breaks eleven rounds of the MARS core with 2^{65} chosen plaintexts, 2^{70} memory, and 2^{229} partial decryptions. Our attack breaks eight rounds of Serpent with 2^{114} chosen plaintexts, 2^{119} memory, and 2^{179} partial decryptions.

1 Introduction

MARS [BCD+98] and Serpent [ABK98] are block ciphers that have been proposed as AES candidates [NIST97a,NIST97b]. More recently, both were chosen as AES finalists. We have spent considerable time in the last few months cryptanalyzing both ciphers, with the bulk of our results appearing in [KS00,KKS00]. During our work on MARS, we developed a new class of attack based on David Wagner's boomerang and inside-out attacks [Wag99]. In this paper, we present this new class of attack, first in the abstract sense, and then in terms of specific attacks on reduced-round variants of the MARS core and of Serpent.

The MARS core provides an excellent target for these attacks. We know of no good iterative differential characteristics, nor of any good differentials of any useful length. However, there is a three-round characteristic and a three-round truncated differential each with probability one. Since these attacks allow concatenation of short differentials that don't connect in the normal sense of differential attacks, they are quite useful against the MARS core. Similarly, Serpent provides an excellent target for these attacks, because the main problem in mounting a differential attack on Serpent is keeping the differential characteristics used from spreading out to large numbers of active S-boxes; using boomerangs, amplified boomerangs, and related ideas, we can make use of differentials with relatively few active S-boxes, and connect them using the boomerang construction.

⋆ Part of this work was done while working for Counterpane Internet Security, Inc.

The underlying "trick" of the boomerang attack is to mount a differential attack with first-order differentials that don't normally connect through the cipher; these first-order differentials are connected by a second-order differential relationship in the middle of the cipher, by way of some adaptive-chosen-ciphertext queries. The underlying "trick" of the boomerang-amplifier attack is to use large numbers of chosen plaintext pairs to get that second-order differential relationship to appear in the middle of the cipher by chance. Extensions allow us to use structures of structures to get the second-order differential relationship in the middle for many pairs of texts at once.

1.1 Impact of the Results

The most important impact of our results is the introduction of a new cryptanalytic technique. This technique belongs to the same general class as the boomerang and miss-in-the-middle attacks; the attacker builds structures of a certain kind from right pairs for differentials through part of the cipher. Other members of this class of attacks are known to us, and research is ongoing into their uses and limitations.

Additionally, we provide the best known attack on the MARS core, breaking up to eleven rounds faster then brute-force search. This attack does not threaten full MARS; even if the cryptographic core had only eleven rounds, we know of no way to mount an attack on the core through the key addition/subtraction and unkeyed mixing layers present in the full MARS.

We also demonstrate this powerful new attack on a reduced-round variant of Serpent. Again, this attack does not threaten the full 32-round Serpent.

The attacks described in this paper are summarized below. However, these specific attacks are not the focus of the paper; instead, the focus is the new cryptanalytic technique.

Summary of Results

Cipher (rounds)	**Texts** (chosen plaintexts)	**Memory** (bytes)	**Work** (decryptions)
MARS Core (11)	2^{65}	2^{69}	2^{229} partial
Serpent (8)	2^{114}	2^{119}	2^{179} 8-round

2 Boomerangs, Inside-Out Attacks, and the Boomerang-Amplifier

2.1 Preliminaries

In [Wag99], Wagner introduces two new attacks: the boomerang attack and the inside-out attack. An understanding of both attacks is necessary to understand our new attack. In order to build on these concepts later, we briefly review the concepts from [Wag99].

Most of the attacks in this section make use of a block cipher E composed of two halves, e_0, e_1. That is, $E(X) = e_0(e_1(X))$. We also use the following notation to describe plaintext i as it is encrypted under E:

$$\begin{aligned} X_i &\leftarrow \text{ plaintext} \\ Y_i &\leftarrow e_0(X_i) \\ Z_i &\leftarrow e_1(Y_i) \text{ (ciphertext)} \end{aligned}$$

An important side-note: A normal differential always has the same probability through a cipher (or subset of cipher rounds) going forward and backward; by contrast a truncated differential can have different probabilities going forward and backward.

2.2 The Inside-Out Attack

Consider a situation in which we have probability one truncated differentials through both e_1 and through e_0^{-1}, both with the same starting difference. That is, we have

$$\begin{aligned} \Delta_0 &\to \Delta_1 \text{ through } e_1 \\ \Delta_0 &\to \Delta_2 \text{ through } e_0^{-1} \end{aligned}$$

In this case, we can mount an attack to distinguish E from a random permutation as follows:

1. Observe enough known plaintext/ciphertexts pairs that we expect R pairs of texts with the required difference in the middle. That is, we expect about R pairs (i, j) for which $Y_i \oplus Y_j = \Delta_0$.
2. Identify the pairs of inputs where $X_i \oplus X_j = \Delta_2$.
3. Identify the pairs of outputs where $Z_i \oplus Z_j = \Delta_1$.
4. Count the number of pairs that overlap (that is, the pairs that are right pairs in both input and output); if this count is substantially higher than would be expected from a random permutation, we distinguish E from a random permutation.

Suppose we had the following probabilities for a random i, j pair:

$$\begin{aligned} Pr[X_i \oplus X_j = \Delta_2] &= p_0 \\ Pr[Z_i \oplus Z_j = \Delta_1] &= p_1 \end{aligned}$$

That is, the probability of a randomly selected pair fitting the truncated difference Δ_2 is p_0, and its probability of fitting Δ_1 is p_1. In this case, we have:

$N =$ Number of total plaintext/ciphertext pairs.

$N_0 = N * p_0 =$ Expected number of counted right input pairs for random perm.

$N_1 = N * p_0 * p_1 =$ Expected number of those right input pairs counted as right output pairs.

The number of pairs that are right pairs for both input and output is then binomially distributed, and can be approximated with a normal distribution with $\mu = N_1$ and $\sigma \approx \sqrt{N_1}$. When we have a right pair in the middle (that is, when $Y_i \oplus Y_j = \Delta_0$), then i, j must be a right pair for both inputs and outputs. We expect R right pairs in the middle; that means that for E, we expect about $N_1 + R$ right pairs for both input and output, while for a random permutation, we expect only about N_1. When R is much larger than $\sqrt{N_1}$, this becomes detectable with reasonably high probability. (For very low probabilities, we can use $R > 16\sqrt{N_1}$, which has an astronomically low probability.)

This gives us a way to attack a cipher even when there are no good differentials through the whole cipher, by waiting for the required difference to occur at random in the middle of the cipher. This idea can be extended to deal with differentials with lower probabilities; see [Wag99] for details.

2.3 Boomerangs

Another fundamental idea required to understand the boomerang-amplifier attack is the boomerang attack. Consider the same cipher $E(X) = e_1(e_0(X))$, but now suppose that there are excellent differentials through e_0, e_0^{-1}, and e_1^{-1}. For this discussion, we assume that these are normal differentials and that they have probability one. The attack works with lower-probability differentials, and with truncated differentials; for a full discussion of the additional complexities these raise, see [Wag99].

We thus have the following differentials:

$$\Delta_0 \rightarrow \Delta_1 \text{ through } e_0 \text{ and } e_1$$

with probability one. Now, despite the fact that $\Delta_1 \neq \Delta_0$ and despite a lack of any high-probability differential through E, we can still distinguish E from a random permutation as follows:

1. Request a right pair for e_0 as input, X_0, X_1, s.t. $X_0 \oplus X_1 = \Delta_0$.
2. After e_0, these have been encrypted to Y_0, Y_1, and have the relationship $Y_0 \oplus Y_1 = \Delta_1$. After e_1, these have been encrypted to Z_0, Z_1, with no predictable differential relationship.
3. We make two right pairs for e_1^{-1} from this pair, by requesting the decryption of $Z_2 = Z_0 \oplus \Delta_1$ and $Z_3 = Z_1 \oplus \Delta_1$.
4. Z_2, Z_3 are decrypted to Y_2, Y_3, with the relationships $Y_2 \oplus Y_0 = \Delta_0$ and $Y_3 \oplus Y_1 = \Delta_0$.
5. This determines the differential relationship between Y_2 and Y_3.

$$\begin{aligned} Y_0 \oplus Y_1 = \Delta_1; Y_0 \oplus Y_2 = \Delta_0; Y_1 \oplus Y_3 = \Delta_0 \\ \text{thus: } Y_2 \oplus Y_3 = \Delta_1 \end{aligned}$$

6. Because $Y_2 \oplus Y_3 = \Delta_1$, we have a right output pair for e_0. Since we're dealing with a normal differential, we know that the differential must go the other direction, so that $X_2 \oplus X_3 = \Delta_0$.

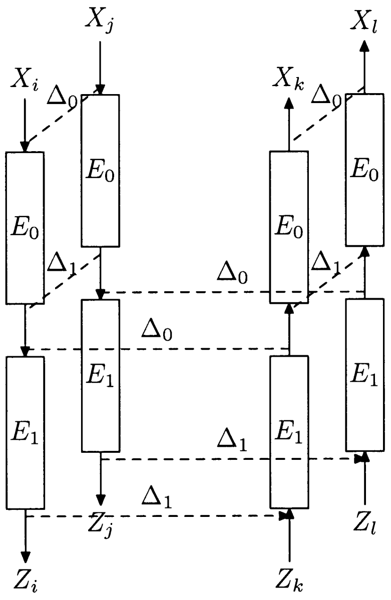


Fig. 1. The Boomerang Attack

7. A single instance of this working for a random permutation has probability 2^{-128} for a 128-bit block cipher, so this can be used very effectively to distinguish E from a random permutation.

Note that the really powerful thing about this attack is that it allows a differential type attack to work against a cipher for which there is no good differential through the whole cipher.

2.4 Turning the Boomerang into a Chosen-Plaintext Attack

We can combine the ideas of the inside-out and boomerang attacks to turn the boomerang attack into an attack that requires only chosen plaintext queries; unlike the boomerang attack, this new attack does not require adaptive-chosen-ciphertext queries. Note that we're dealing with the same cipher $E(X) = e_1(e_0(X))$.

Suppose we are dealing with a 128-bit block cipher, and are thus dealing with 128-bit differences. We request 2^{65} random chosen plaintext pairs X_{2i}, X_{2i+1} such that $X_{2i} \oplus X_{2i+1} = \varDelta_0$. Since we are dealing with probability one differentials, this gives us 2^{65} pairs Y_{2i}, Y_{2i+1} such that $Y_{2i} \oplus Y_{2i+1} = \varDelta_1$. We expect about two pairs (i, j) for which $Y_{2i} \oplus Y_{2j} = \varDelta_0$. When we have an i, j pair of this kind

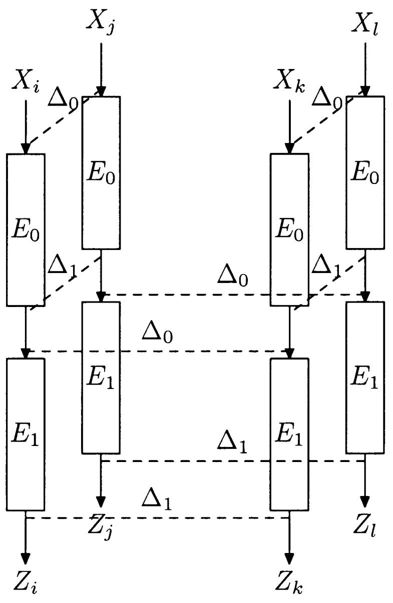


Fig. 2. The Boomerang-Amplifier Attack

we have the boomerang property:

$$Y_{2i} \oplus Y_{2i+1} = \Delta_1; Y_{2j} \oplus Y_{2j+1} = \Delta_1; Y_{2i} \oplus Y_{2j} = \Delta_0$$
$$\text{thus: } Y_{2i+1} \oplus Y_{2j+1} = \Delta_0$$
$$\text{and so: } Z_{2i} \oplus Z_{2j} = \Delta_1; Z_{2i+1} \oplus Z_{2j+1} = \Delta_1$$

There are about 2^{129} possible i, j pairs. The probability that any given pair will satisfy the last two equations is 2^{-256}. We can thus use the above technique to distinguish E from a random permutation.

We call this attack a boomerang-amplifier attack, because the boomerang structure "amplifies" the effect of a low-probability event ($Y_{2i} \oplus Y_{2j} = \Delta_0$) enough that it can be easily detected. By contrast, the inside-out attack amplifies such a low-probability event by detecting a signal from both input and output of the cipher.

2.5 Comparing Boomerangs and Boomerang Amplifiers

It is worthwhile to compare boomerang-amplifiers with the original boomerangs, in terms of attacks made possible. All else being equal, boomerangs require far fewer total queries than boomerang amplifiers, because in a boomerang-amplifier attack, we have to request enough right input pairs to expect the internal collision property that allows us to get the desired relationship between the pairs. Thus, in the example above, we're trading off 2^{65} chosen plaintext queries for two adaptive chosen ciphertext queries.

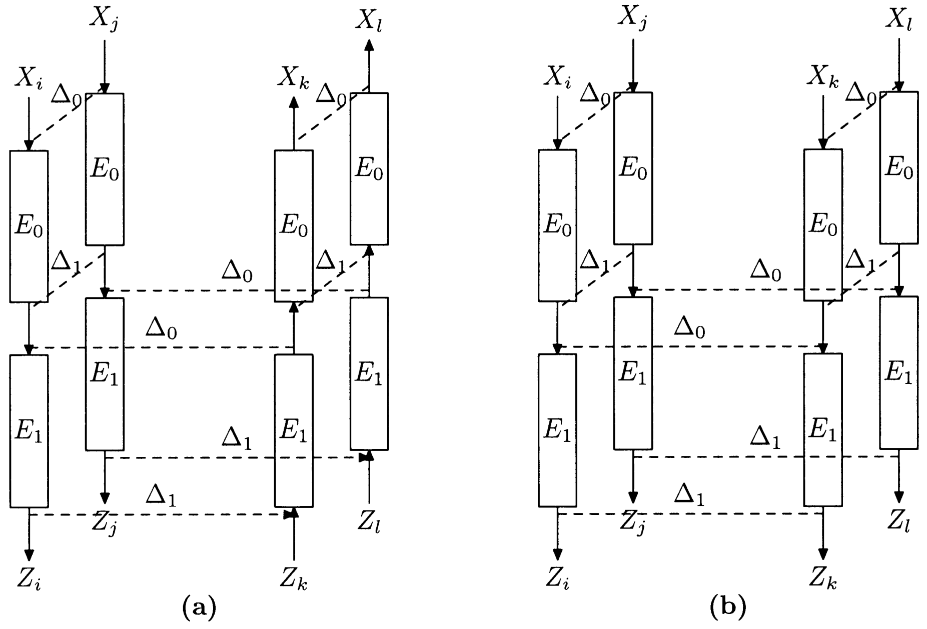


Fig. 3. Comparing Boomerangs and Boomerang-Amplifiers; Note Direction of Arrows

This might not look all that useful. However, there are three things that make this useful in many attacks:

1. When mounting an attack, we often need to guess key material on one end or the other of the cipher. With a chosen-plaintext/adaptive chosen-ciphertext attack model, we must increase our number of requested plaintexts/ciphertexts when we have to guess key material on either end. With a chosen-plaintext only attack, we can guess key material at the end of the cipher, and not have to increase our number of chosen plaintexts requested.
2. We can use the boomerang-amplifier, not just on pairs, but on k-tuples of texts.
3. We can use the boomerang-amplifier to get pairs (or k-tuples) of texts though part of the cipher, and then cover the remaining rounds of the cipher with truncated differentials or differential-linear characteristics. In this way, we can use truncated differentials that specify only a small part of the block, and couldn't be used with a standard boomerang attack.

2.6 Boomerang-Amplifiers with 3-Tuples

Consider a method to send 3-tuples of texts through e_0 with the property that when $X_{0,1,2}$ are chosen properly, Y_i, Y_i^*, Y_i^{**} have some simple XOR relationship, such as $Y_i^* = Y_i \oplus t^*$ and $Y_i^{**} = Y_i \oplus t^{**}$. We can carry out a boomerang-amplifier attack using these 3-tuples. Consider:

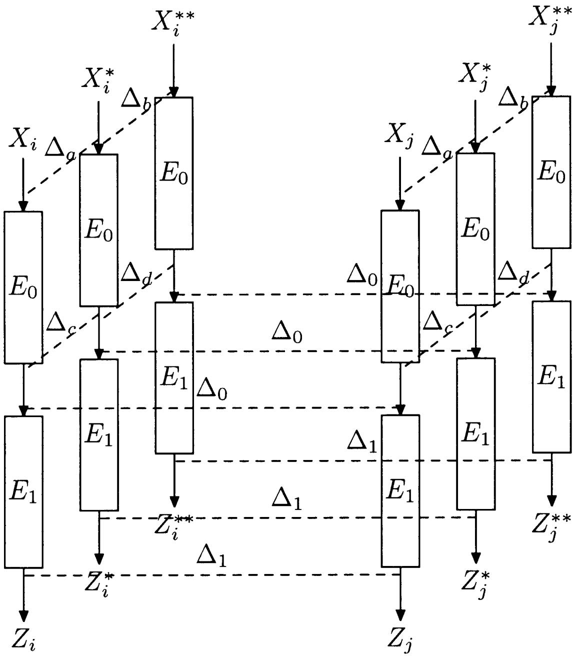


Fig. 4. A Boomerang-Amplifier with 3-Tuples

1. Request 2^{65} such 3-tuples, X_i, X_i^*, X_i^{**}.
2. We expect about one instance where $Y_i \oplus Y_j = \Delta_0$, by the birthday paradox.
3. For that instance, we get *three* right pairs through the cipher:

$$(Z_i, Z_j); (Z_i^*, Z_j^*); (Z_i^{**}, Z_j^{**})$$

This happens because:

$$Y_i \oplus Y_j = \Delta_0; Y_i^* = Y_i \oplus t^*; Y_j^* = Y_j \oplus t^*$$
$$\text{Thus: } Y_i^* \oplus Y_j^* = Y_i \oplus Y_j \oplus t^* \oplus t^* = Y_i \oplus Y_j = \Delta_0$$

The same idea works in a boomerang attack, but is of no apparent use. However, in a boomerang-amplifier attack, we are able to use this trick to get through more rounds. Because we're doing a chosen-plaintext attack, we can look for patterns that are apparent from k-tuples of right pairs, even through several more rounds of the cipher. Conceptually, we can use this to increase the "amplification" on the attack.

2.7 Detecting the Effects of the Boomerang-Amplifiers

A boomerang 4-tuple $(X_{i,j,k,l}, Z_{i,j,k,l})$ has the property that $X_{i,j}$ and $X_{k,l}$ are right input pairs, and $Z_{i,k}$ and $Z_{j,l}$ are right output pairs. When we mount a boomerang-amplifier attack, we know that $X_{i,j}$ and $X_{k,l}$ are right pairs, because we have chosen them to be right pairs. We thus detect a right pair of pairs by noting that $Z_{i,k}$ and $Z_{j,l}$ are right output pairs.

There is a straightforward trick for speeding up searches for these right pairs of pairs among large sets of pairs. It is easy to see that

$$\begin{aligned} Z_i \oplus Z_k &= \Delta_2 \\ Z_j \oplus Z_l &= \Delta_2 \\ Z_i \oplus Z_j &= Z_i \oplus Z_l \oplus \Delta_2 \\ &= Z_k \oplus Z_l \end{aligned}$$

This means that we can build a sorted list of the output pairs from a large set of input pairs, in which each entry in the list contains the XOR of the ciphertexts from one pair. When both $Z_{i,k}$ and $Z_{j,l}$ are right output pairs, then $Z_i \oplus Z_j = Z_k \oplus Z_l$. A variant of this technique works even when the output differences are truncated differences; in that case, the differential relationship works only in the fixed bits of the difference.

When we apply boomerang-amplifiers to larger blocks of texts, we must find other ways to detect them. For example, below we describe an attack in which we concatenate a differential-linear characteristic with probability one to the differences after several rounds resulting from right pairs in the middle. This means that each right pair in the middle gives us one bit that has to take on a certain value; we request batches of over 256 entries, and look for right pairs of batches. Each batch of N texts results, as ciphertext, in an N-bit string. We build a sorted list of these strings, and find the matches, which must come from right pairs of batches.

Alternatively, we can simply build sorted lists of all the Z_i and all the $Z_i \oplus \Delta_1$, and then look for matches.

3 Amplified Boomerangs and the MARS Core

3.1 The MARS Core

MARS [BCD+98] is a heterogenous, target-heavy, unbalanced Feistel network (to use the nomenclature from [SK96]). At the center are 16 core rounds: eight forward core rounds and eight backward core rounds. Surrounding those rounds are 16 keyless mixing rounds: eight forward mixing rounds before the core, and eight backward mixing rounds after the core. Surrounding that are two whitening rounds: one at the beginning of the cipher and another at the end.

The design of MARS is fundamentally new; the whitening and unkeyed mixing rounds jacket the cryptographic core rounds, making any attacks on the core rounds more difficult. The core rounds, by contrast, must provide the underlying cryptographic strength, by giving a reasonable approximation of a random permutation family. This can be seen by considering the fact that if the cryptographic core is replaced by a known random permutation, there is a trivial meet-in-the-middle attack. Related results can be found in [KS00]. It thus makes sense to consider the strength of the MARS core rounds alone, in order to try to evaluate the ultimate strength of MARS against various kinds of attack.

Both forward and backward core rounds use the same E function, which takes one 32-bit input and two subkey words, and provides three 32-bit words. The only difference between forward and backward rounds is the order in which the outputs are combined with the words. For more details of the MARS core rounds, see [BCD+98,KS00].

Notation and Conventions. The notation we use here for considering the MARS core rounds differs from the notation used in [BCD+98]. One forward core round may be represented as follows:

1. $(A_{i-1}, B_{i-1}, C_{i-1}, D_{i-1})$ are the four 32-bit words input into round i.
2. (A_i, B_i, C_i, D_i) are the four 32-bit output words from round i.
3. The two round keys are K_i^+ and $K_i^\times$.
4. K_i^+ has 32 bits of entropy; $K_i^\times$ has just under 30 bits of entropy, because it is always forced to be congruent to 3 modulo 4. The full round thus has 62 bits of key material.
5. One forward core round may be expressed as:

$$\begin{aligned}
F_i^\times &= ((A_{i-1} \lll 13) \times K_i^\times) \lll 10 \\
F_i^+ &= (A_{i-1} + K_i^+) \lll (F_i^\times \ggg 5) \\
F_i^s &= (S[\text{low nine bits } (A_{i-1} + K_i^+)] \oplus (F_i^\times \lll 5) \oplus F_i^\times) \lll F_i^\times \\
D_i &= A_{i-1} \lll 13 \\
A_i &= B_{i-1} + F_i^s \\
B_i &= C_{i-1} + F_i^+ \\
C_i &= D_{i-1} \oplus F_i^\times
\end{aligned}$$

We find this notation easier to follow than the original MARS paper's notation, and so will use it for the remainder of this paper. On pages 12–13 of the original MARS submission document, these values are referred to as follows:

– F^s is referred to as either $out1$ or L.
– F^+ is referred to as either $out2$ or M.
– $F^\times$ is referred to as either $out3$ or R.
– K^+ is referred to as K.
– $K^\times$ is referred to as K'.

Useful Properties. The MARS core function is difficult to attack for many rounds. However, there are a number of useful properties which we have been able to use to good effect in analyzing the cipher. These include:

1. For $A = 0$, $F^{\times}$ is always zero, F^{s} is S[low nine bits of K^{+}], and F^{+} is always K^{+}.
2. There is a three-round truncated differential $(0, 0, 0, \delta_0) \rightarrow (\delta_1, 0, 0, 0)$ with probability one.
3. The multiply operation can only propagate changes in its input toward its higher-order bits. This leads to a number of probability one linear characteristics for small numbers of rounds, one of which we will use in an attack, below.

Showing Differentials and Truncated Differentials. In the remainder of this section, we will represent differentials as 4-tuples of 32-bit words, such as $(0, 0, 0, 2^{31})$. We will represent truncated differentials in the same way, but with variables replacing differences that are allowed to take on many different values, as in $(0, 0, 0, x)$, which represents the a difference of zero in the first three words of the block, and an unknown but nonzero difference in the last word of the block. Differences within a word will be shown as sequences of known zero or one bits, "dont care" bits, or variables. Thus, to show a word whose high 15 bits are zeros, whose 16th bit may take on either value, and whose low 16 bits we don't care about, we would use $(0^{15}, a, ?^{16})$. If we have a difference in which the first three words are zero, and the last word has its high 15 bits zero, its 16th bit able to take on either value, and with all other bits unimportant for the difference, we would show this as $(0, 0, 0, (0^{15}, a, ?^{16}))$.

Sending a Counter Through Three Rounds of the MARS Core. Here is one additional property of the MARS core that turns out to be very useful: We can send a counter through three rounds of MARS core by choosing our inputs correctly.

Consider a set of inputs $(0, t, u, i)$, where t, u are random 32-bit words, and i is a counter that takes on all 2^{32} possible values. After three rounds, this goes to $(i + v, w, x, y)$, where v, w, x, y are all 32-bit functions of t, u.

When t, u are held constant, we get a set of 2^{32} texts whose A_3 values run through all possible values in sequence. We don't know the specific values, but for any additive difference, δ, we can identify 2^{32} pairs with that difference after three rounds.

Similarly, we can choose a restricted set of i values. For example, if we run i through all values between 0 and 255, we get 128 different pairs with a difference of 128. This will prove useful in later attacks.

In 3.3, the property described here will be exploited to mount a far more powerful attack on the MARS core.

3.2 Attacking MARS with a Simple Amplified Boomerang

The MARS core has a three-round differential characteristic with probability one: $(0, 0, 0, 2^{31}) \rightarrow (2^{31}, 0, 0, 0)$. It also has a three-round truncated differential with probability one: $(0, 0, 0, \alpha) \rightarrow (\beta, 0, 0, 0)$. We can use these two characteristics to mount a boomerang-amplifier attack through six rounds of the cipher.

We request about 2^{48} input pairs X_{2i}, X_{2i+1}, such that $X_{2i} \oplus X_{2i+1} = (0, 0, 0, 2^{31})$. As we described above, these pairs are encrypted to pairs Y_{2i}, Y_{2i+1} such that $Y_{2i} \oplus Y_{2i+1} = (2^{31}, 0, 0, 0)$.

After we have about 2^{48} such pairs, we expect to have one pair (i, j) such that $Y_{2i} \oplus Y_{2j} = (0, 0, 0, \alpha)$ for any $\alpha \neq 0$. For this pair, we can solve for $Y_{2i+1} \oplus Y_{2j+1}$; we get $(0, 0, 0, \alpha)$ in that difference as well.

We thus get two right input pairs after round three, and two right output pairs from round six. Among 2^{48} right input pairs, we have about 2^{95} pairs of right input pairs. Since the probability of randomly getting an output pair with difference $(\beta, 0, 0, 0)$ for any $\beta \neq 0$ is 2^{-96}, and since we expect one 4-tuple with two such output pairs, we will easily distinguish six rounds of MARS core from a random permutation.

3.3 A Boomerang-Amplified Differential-Linear Attack on Eleven Rounds

We can combine the above idea with two other properties of the MARS core to build a much more powerful attack, which is properly classified as either a boomerang-amplified differential-linear attack, or a boomerang-amplified truncated-differential attack. Our attack consists of the following:

1. We choose inputs so that we get batches of 280 texts following the pattern
$$(s, t, u, v), (s + 1, t, u, v), (s + 2, t, u, v), ..., (s + 279, t, u, v)$$
We use the technique described in section 3.1 to do this.
2. We request 2^{57} such batches, so that we can expect a pair of batches with a truncated difference $\delta = (0, 0, 0, (?^{13}, 0^{17}, ?^2))$ between each corresponding pair of texts in the batch. That is, after round three, the *first* elements of one pair of batches have the following relationship:
$$v^* - v = \delta$$
$$s^* - s = t^* - t = u^* - u = 0$$
It follows by simple arithmetic that the ith elements of the pair of batches have difference $(0, 0, 0, \delta)$. Note that this is an *additive* difference.
3. When we get this right pair of batches, we get 280 pairs with this additive difference into the output of round three. This means we get 280 right pairs in the output of round six.
4. We are then able to cover two more rounds with a linear characteristic with $p \approx 1$.

5. We guess our way past the last two full rounds, and past part of the third round from the end. This gives us a part of the output from the eighth round. This requires 62 bits for each full round, and 39 bits for the partial guess, thus a total of 163 bits guessed.
6. For each of the 2^{57} batches, we extract the linear characteristic from the output of the eighth round of each of the 280 texts. We thus get a 280-bit string from each batch, for each partial key guess.
7. For each key guess, we build a sorted list of these 280-bit strings, and find the match. The probability of finding a match of a 280-bit string in 2^{57} texts is about 2^{-167}; we are trying to find a match for 2^{163} different key guesses. Thus, we expect to have no false matches, and we are vanishingly unlikely to have more than a small number of false matches.

Getting the Right Difference Between the Batches. Consider only the first element in each batch. There are 2^{57} such elements. We need one pair such that after round three, it has a truncated difference of $(0, 0, 0, (?^{13}, 0^{17}, ?^{2}))$; that is, with all but 15 of its bits zeros. The probability of a random pair of texts having this difference is 2^{-113}. There are about 2^{113} pairs of these texts, and so we expect this difference to happen about once.

When this difference happens between the first elements of a pair of batches, it is easy to see that it must also occur between the second elements, and the third, and so on. We thus get a right pair of batches, yielding 280 right pairs in the output from round three.

The Linear Characteristic/Truncated Differential. Consider a single pair of texts with the difference $(0, 0, 0, (?^{12}, a, 0^{17}, ?^{2}))$ at the output of round three, where a is a single unknown bit. We care only about bit a in our attack. When we originally developed this attack, we thought in terms of a differential-linear characteristic with probability one. In this case, this is equivalent to a truncated differential with only one bit specified. Here, we describe this in terms of the truncated differential attack.

First, we note that with probability of very nearly one $(1 - 2^{-17})$, a will be unchanged for any given pair of corresponding texts after round six, in the output truncated difference $((?^{12}, a, ?^{19}), 0, 0, 0)$. The probability that this doesn't change for any corresponding pair of texts in the right pair of batches is about 0.998. (The number of times a changes is binomially distributed, with $n = 280, p = 2^{-17}$.)

In the seventh round, bit a is rotated to the low-order bit input into the multiply operation; this means that bit ten of $F_7^{\times}$ is a. This is a "backward" core round, so the output from the seventh round leaves us with truncated difference $((?^{21}, a, ?^{10}), ?, ?, ?)$. In the next round, the leftmost word is changed only by being rotated. We thus get the following truncated difference in the output from the eighth round: $(?, ?, ?, (?^{7}, a, ?^{23}))$.

This single bit appears as a constant in the differences for all 280 corresponding pairs of the right pair of batches.

Guessing Key Material. We guess the low nine bits of K_9^+, and the full 30-bit $K_9^\times$. This allows us to backtrack through f_9^s, and thus to recover bit a for all the texts. We then guess our way past rounds ten and eleven by guessing 62 bits for each round.

Required Resources for the Attack. We attack eleven rounds of MARS core; five forward and six backward.

We request 2^{57} batches of 280 texts each, and thus must request a total of about 2^{65} chosen plaintexts. We get a batch of 2^{65} ciphertexts back out, which we must store and then examine to mount our attack.

We must guess our way past two full MARS core rounds (at a cost of 62 bits each), plus part of a third MARS core round (at a cost of 39 bits). We thus must guess a total of 163 bits. We must try 2^{163} times to find a match, once per key guess.

For each key guess, we have to do the following steps:

1. Do the partial decryption on about 2^{65} ciphertexts, and extract one bit per ciphertext, at a cost of about 2^{65} partial decryptions.
2. Arrange the resulting bits as 2^{57} 280-bit strings.
3. Sort the 280-bit strings, at a cost of about $57 \times 2^{57} \approx 2^{63}$ swapping operations' work.

To simplify our analysis, we assume that the work of doing the 2^{65} partial decryptions dominates the work of sorting the 280-bit strings; we thus require about $2^{163} \times 2^{65} = 2^{228}$ partial decryptions' work to mount the attack.

The total memory required for the attack is 2^{70} bytes, sufficient for 2^{66} ciphertexts.

4 Boomerang-Amplifiers and Serpent

Serpent is a 32-round AES-candidate block cipher proposed by Ross Anderson, Eli Biham, and Lars Knudsen [ABK98]. In this section we show how one can apply the amplified boomerang technique to reduced-round Serpent variants. Additional attacks against reduced-round Serpent can be found in [KKS00]. Unlike those used in MARS, the differentials used in our attacks on Serpent do not have probability one.

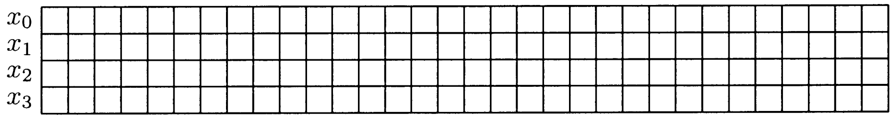
4.1 Description of Serpent

Serpent is a 32-round block cipher operating on 128-bit blocks. The Serpent design documentation describes two versions of Serpent: a bitsliced version and a non-bitsliced version. Both versions are functionally equivalent. The difference between the bitsliced and non-bitsliced versions of Serpent is the way in which the data is represented internally. In this document we shall only consider the bitsliced version of Serpent.

Let B_i represent Serpent's intermediate state prior to the ith round of encryption; B_0 is the plaintext and B_{32} is the ciphertext. Let K_i represent the 128 bit ith round subkey and let S_i represent the application of the ith round S-box. Let L represent Serpent's linear transformation (see [ABK98] for details). Then the Serpent round function is defined as:

$$\begin{aligned} X_i &\leftarrow B_i \oplus K_i \\ Y_i &\leftarrow S_i(X_i) \\ B_{i+1} &\leftarrow L(Y_i) \qquad i = 0, \ldots, 30 \\ B_{i+1} &\leftarrow Y_i \oplus K_{i+1} \; i = 31 \end{aligned}$$

In the bitsliced version of Serpent, one can consider each 128-bit block X_i as the concatenation of four 32-bit words x_0, x_1, x_2, and x_3. Pictorially, one can represent Serpent's internal state X_i using diagrams such as the following: Serpent uses eight S-boxes S_i where the indices i are reduced modulo 8; e.g.,



$S_0 = S_8 = S_{16} = S_{24}$. Each S-box takes four input bits and produces four output bits. The input and output nibbles of the S-boxes correspond to the columns in the preceding diagram (where the most significant bits of the nibbles are the bits in the word x_3).

We use X' to represent an XOR difference between two values X and X^*.

4.2 Distinguishing Seven Rounds of Serpent

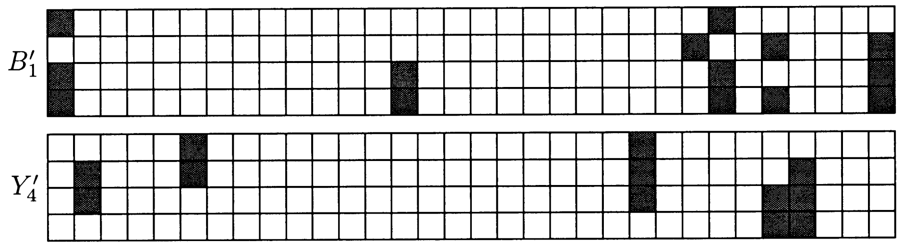
Let us consider a seven-round Serpent variant $E_1 \circ E_0$ where E_0 corresponds to rounds one through four of Serpent and E_1 corresponds to rounds five through seven of Serpent.

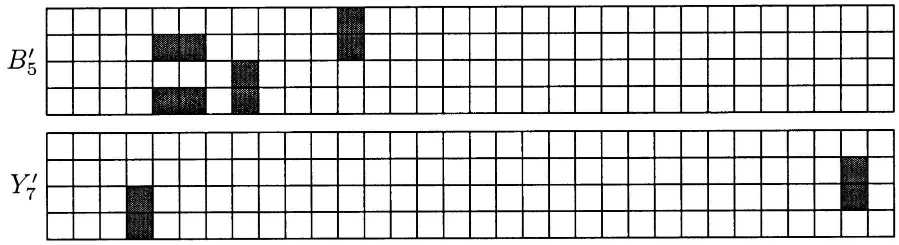
There are several relatively high-probability characteristics through both halves of this seven round Serpent variant. Let us consider two such characteristics $B'_1 \rightarrow Y'_4$

and $B'_5 \rightarrow Y'_7$

where $B'_1 \rightarrow Y'_4$ is a four-round characteristic through E_0 with probability 2^{-31} and $B'_5 \rightarrow Y'_7$ is a three-round characteristic through E_1 with probability 2^{-16}. Additional information on these characteristics can be found in [KKS00].

We can immediately combine these two characteristics to form a seven round boomerang distinguishing attack requiring 2^{95} chosen plaintext queries and 2^{95} adaptive chosen ciphertext queries. Using the amplified boomerang technique, however, we can construct a chosen-plaintext only distinguishing attack requiring 2^{113} chosen plaintext queries.





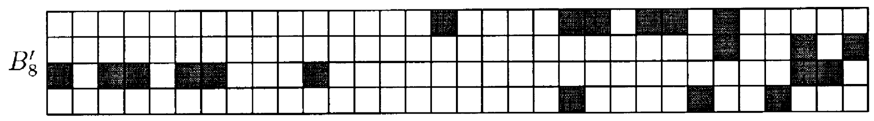
The details of the attack are as follows. We request 2^{112} plaintext pairs with our input difference B'_1. After encrypting with the first half of the cipher E_0, we expect roughly 2^{81} pairs to satisfy the first characteristic $B'_1 \rightarrow Y'_4$. There are approximately 2^{161} ways to form quartets using these 2^{81} pairs. We expect there to be approximately 2^{33} quartets (Y_4^0, Y_4^1) and (Y_4^2, Y_4^3) such that $Y_4^0 \oplus Y_4^2 = L^{-1}(B'_5)$. However, because (Y_4^0, Y_4^1) and (Y_4^2, Y_4^3) are right pairs for the first half of the cipher, and $Y_4^0 \oplus Y_4^1 = Y_4^2 \oplus Y_4^3 = Y'_4$, we have that $Y_4^1 \oplus Y_4^3$ must also equal $L^{-1}(B'_5)$. In effect, the randomly occurring difference between Y_4^0 and Y_4^2 has been "amplified" to include Y_4^1 and Y_4^3.

At the input to E_1 we expect approximately 2^{33} quartets with a difference of (B'_5, B'_5) between the pairs. This gives us approximately two quartets after the seventh round with an output difference of (Y'_7, Y'_7) across the pairs. We can identify these quartets by intelligently hashing our original ciphertext pairs with our ciphertext pairs XORed with (Y'_7, Y'_7) and noting those pairs that collide. In a random distribution, the probability of observing a single occurrence of the cross-pair difference (Y'_7, Y'_7) is approximately 2^{-33}.

4.3 Eight-Round Serpent Key Recovery Attack

We can extend the previous distinguishing attack to an eight-round key-recovery attack requiring 2^{113} chosen plaintext pairs, 2^{119} bytes of memory, and work equivalent to approximately 2^{179} eight-round Serpent encryptions. This attack covers rounds one through eight of Serpent. If we apply the linear transformation

L to Y_7' we get the difference:



Given 2^{113} chosen plaintext pairs with our input difference B_1', we expect approximately eight pairs of pairs with cross-pair difference (Y_7', Y_7') after the seventh round. This corresponds to eight pairs of pairs with difference (B_8', B_8') entering the eighth round. By guessing 68 bits of Serpent's last round subkey K_9, we can peel off the last round and perform our previous distinguishing attack.

5 Conclusions

In this paper, we have introduced a new kind of attack that is closely related to the boomerang attack of Wagner [Wag99]. We have applied this attack to reduced-round versions of both the MARS core and of Serpent.

5.1 Related Attacks

There is a set of related attacks, including the miss-in-the-middle, boomerang, and amplified boomerang attacks, which deal with pairs of differentials that reach to the same point in the intermediate state of the cipher, but which don't connect as needed for conventional attacks. A miss-in-the-middle attack gives us three texts in the middle that *can't* fit a certain second-order differential relationship. A boomerang or amplified boomerang gives us a 4-tuple that does fit a certain second-order differential relationship. However, these are different from standard higher-order differential attacks in that the second-order differential relationship doesn't continue to exist through multiple rounds. Instead, this relationship serves only to connect pairs of texts with a first-order differential relationship in the middle of the cipher.

In some sense, this is similar to the way structures are used with higher-order differential relationships, in order to use first-order differentials more efficiently. Thus, we might have two good differentials through the first round that will get us to our desired input difference:

$$\Delta_0 \to \Delta_1$$
$$\Delta_2 \to \Delta_1$$

It's a common trick to request $X, X \oplus \Delta_0, X \oplus \Delta_2, X \oplus \Delta_0 \oplus \Delta_2$, which will give us four right pairs, two for each differential, for the price of only four texts. We're requesting a 4-tuple of texts with a second-order differential relationship, but it doesn't propagate past the first round. This is a second-order differential attack in exactly the same sense as the boomerang and amplified boomerang attacks are second-order differential attacks.

The boomerang and boomerang-amplifier attacks are, in some sense, a new way of building structures of texts *inside* the middle of the cipher. These structures have a second order relationship, which allows the four texts to take part in four right pairs. However, in the boomerang and boomerang-amplifier attacks, two of the right pairs go through the first half of the cipher, and two go through the second half of the cipher.

5.2 Applying the Attack to Other Algorithms

We have not yet applied this attack to other algorithms besides MARS and Serpent. However, there is a common thread to situations in which the attack works: We need to be able to get through many rounds with some differential that has reasonably high probability. In the case of the MARS core, there are probability one differentials for three rounds, simply due to the structure of the cipher. In the case of Serpent, the probability of a differential characteristic is primarily a function of the number of S-boxes in which the difference is active across all rounds of the characteristic. Differences spread out, so that it is possible to find reasonably good characteristics for three or four rounds at a time, but not for larger numbers of rounds, since by then the differences have spread to include nearly all the S-boxes.

Applying this general class of attack to other ciphers will be the subject of ongoing research.

It is worth repeating, however, that this technique does not endanger either Serpent or MARS. In the case of MARS, the cryptographic core is jacketed with additional unkeyed mixing and key addition/subtraction layers, which would make chosen-plaintext attacks like this one enormously more expensive (more expensive than exhaustive search), even if our attack worked against the full cryptographic core. In the case of Serpent, the large number of rounds prevents our attack from working against the full cipher.

Acknowledgements. The "extended Twofish team" met for two week-long cryptanalysis retreats during Fall 1999, once in San Jose and again in San Diego. This paper is a result of those collaborations. Our analysis of MARS and Serpent has very much been a team effort, with everybody commenting on all aspects. The authors would like to thank Niels Ferguson, Mike Stay, David Wagner, and Doug Whiting for useful conversations and comments on these attacks, and for the great time we had together. The authors also wish to thank the reviewers, for useful comments and suggestions, and Beth Friedman, for copyediting the final paper.

References

ABK98. R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," NIST AES Proposal, Jun 1998.

BCD+98. C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S.M. Matyas, L. O'Connor, M. Peyravian, D. Safford, and N. Zunic, "MARS — A Candidate Cipher for AES," NIST AES Proposal, Jun 98.

BS93. E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.

Knu95b. L.R. Knudsen, "Truncated and Higher Order Differentials," *Fast Software Encryption, 2nd International Workshop Proceedings*, Springer-Verlag, 1995, pp. 196–211.

KS00. J. Kelsey and B. Schneier, "MARS Attacks! Cryptanalyzing Reduced-Round Variants of MARS," *Third AES Candidate Conference*, to appear.

KKS00. T. Kohno, J. Kelsey, and B. Schneier, "Preliminary Cryptanalysis of Reduced-Round Serpent," *Third AES Candidate Conference*, to appear.

LH94. S. Langford and M. Hellman, "Differential-Linear Cryptanalysis," *Advances in Cryptology — CRYPTO '94*, Springer-Verlag, 1994.

Mat94. M. Matsui, "Linear Cryptanalysis Method for DES Cipher," *Advances in Cryptology — EUROCRYPT '93 Proceedings*, Springer-Verlag, 1994, pp. 386–397.

NIST97a. National Institute of Standards and Technology, "Announcing Development of a Federal Information Standard for Advanced Encryption Standard," *Federal Register*, v. 62, n. 1, 2 Jan 1997, pp. 93–94.

NIST97b. National Institute of Standards and Technology, "Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES)," *Federal Register*, v. 62, n. 117, 12 Sep 1997, pp. 48051–48058.

SK96. B. Schneier and J. Kelsey, "Unbalanced Feistel Networks and Block Cipher Design," *Fast Software Encryption, 3rd International Workshop Proceedings*, Springer-Verlag, 1996, pp. 121–144.

Wag99. D. Wagner, "The Boomerang Attack," *Fast Software Encryption, 6th International Workshop*, Springer-Verlag, 1999, pp. 156–170.

Correlations in RC6 with a Reduced Number of Rounds

Lars R. Knudsen[1] and Willi Meier[2]

[1] Department of Informatics,
University of Bergen, N-5020 Bergen
[2] FH-Aargau, CH-5210 Windisch

Abstract. In this paper the block cipher RC6 is analysed. RC6 is submitted as a candidate for the Advanced Encryption Standard, and is one of five finalists. It has 128-bit blocks and supports keys of 128, 192 and 256 bits, and is an iterated 20-round block cipher. Here it is shown that versions of RC6 with 128-bit blocks can be distinguished from a random permutation with up to 15 rounds; for some weak keys up to 17 rounds. Moreover, with an increased effort key-recovery attacks faster than exhaustive key search can be mounted on RC6 with up to 12 rounds for 128 bit keys, on 14 rounds for 192 bit keys and on 15 rounds for 256 bit keys.
Keywords. Cryptanalysis. Block Cipher. Advanced Encryption Standard. RC6.

1 Introduction

RC6 is a candidate block cipher submitted to NIST for consideration as the Advanced Encryption Standard (AES). RC6 (see [12]) is an evolutionary development of RC5. Like RC5, RC6 makes essential use of data-dependent rotations. New features of RC6 include the use of four working registers instead of two, and the inclusion of integer multiplication as an additional primitive operation. RC6 is a parameterized family of encryption algorithms, where RC6-$w/r/b$ is the version with word size w in bits, with r rounds and with an encryption key of b bytes.

The AES submission is the version with $w = 32$, $r = 20$, and RC6 is a shorthand notation for this version, whereby the key length can be $b = 16, 24,$ and 32 bytes, respectively. In [4,5] the security of RC6 has been evaluated with respect to differential and linear cryptanalysis. It was concluded that RC6 is secure with respect to differential cryptanalysis for 12 or more rounds. For linear cryptanalysis, some variants are considered in [4]. It was found that a two-round iterative linear approximation leads to the most effective basic linear attack applicable up to 13 rounds. However, no specific method for key-recovery was given. Furthermore, in [4] some potential enhancements of linear attacks using multiple approximations and linear hulls are sketched, and it is estimated that 16 rounds of RC6 can be attacked using about 2^{119} known plaintexts. These

```
Input:      Plaintext stored in four $w$-bit registers $A, B, C, D$
            Number $r$ of rounds
            $w$-bit round keys $S[0], ..., S[2r+3]$

Output:     Ciphertext stored in $A, B, C, D$

Procedure:  $B = B + S[0]$
            $D = D + S[1]$
            for $i = 1$ to $r$ do
            {
              $t = (B \times (2B + 1)) << \lg w$
              $u = (D \times (2D + 1)) << \lg w$
              $A = ((A \oplus t) << u) + S[2i]$
              $C = ((C \oplus u) << t) + S[2i + 1]$
              $(A, B, C, D) = (B, C, D, A)$
            }
            $A = A + S[2r + 2]$
            $C = C + S[2r + 3]$
```

Fig. 1. Encryption with RC6-$w/r/b$.

additional considerations on linear cryptanalysis were used to set a suitable number of rounds for RC6 to be $r = 20$.

In this paper we investigate two-round iterations which are quite different from those considered in [4]. Instead of tracing bitwise linear approximations, we consider input-output dependencies by fixing the least significant five bits in the first and third words of the input block. The correlations of the corresponding two 5-bit integer values at the output are caused by specific rotation amounts in the data dependent rotations and can be effectively measured by χ^2 tests. As confirmed by extensive experiments, this leads to an efficient statistical analysis which considerably improves over the basic linear attack. Estimates of the complexity of our analysis imply that reduced round versions of RC6 with up to 15 rounds are not random.

The linear attacks in [4] deal with correlations between input and output bits, but they do not involve key bits, whereas our statistical analysis can be used to develop a method to find all round subkeys.

This attack is faster than an exhautive key search for the 128-bit version of RC6 with up to 12 rounds, and for the 192-bit and 256-bit versions of RC6 with up to 14 and 15 rounds.

After completion of the first report of this work [9], our attention was drawn to an earlier result by Baudron et al in [1] where an attack similar to ours is outlined. (See also [6].) These results have since been written up in [2].

In the following we briefly recall the description of RC6, see Figure 1.

For a detailed description we refer to [12]. The user-key has length b bytes and the $4w$-bit plaintext block is loaded into words A, B, C, D. These four w-bit words also contain the ciphertext at the end. The key-schedule (see [12]) expands

the user-key into subkeys $S[0], S[1], ..., S[2r+3]$. In our considerations we shall not make use of the detailed description of the key-schedule, but we assume the subkeys to be uniformly random. To describe the encryption algorithm the following notation is used: $(A, B, C, D) = (B, C, D, A)$ means the parallel assignment of values on the right to registers on the left. Moreover, $a \times b$ denotes integer multiplication modulo 2^w, $a << \lg w$ means fixed rotation of the w-bit word a by $\lg w$, the base-two logarithm of w, and $a << b$ denotes rotation of a to the left by the amount given by the least significant $\lg w$ bits of b.
This paper is organized as follows: In section 2 we review χ^2 tests as a useful tool to detect nonuniformness in probability distributions. In Section 3 the relationship between small rotation amounts and correlation in RC6 is investigated and a class of weak keys is identified. In Section 4 distinguishing and key-recovery attacks are developed, and in Section 5 we draw some conclusions.

2 χ^2 Tests

In this section we recall how to distinguish a random source with unknown probability distribution p_X from a random source with uniform distribution p_U. A common tool for this task is the χ^2 test, which is briefly recalled together with some useful facts (see e.g., [7], [8], [10], [13]). We shall later use χ^2 tests to detect correlation between specific input and output subblocks of r-round RC6.

Let $\mathbf{X} = X_0, X_1, ..., X_{n-1}$ be independent and identically distributed random variables taking values in the set $\{a_0, a_1, ..., a_{m-1}\}$ with unknown probability distribution. Then the χ^2 test is used to decide if an observation $X_0, X_1, ..., X_{n-1}$ is consistent with the hypothesis $Pr\{X = a_j\} = p(j)$ for $0 \leq j < m$, where $p_X = \{p(j)\}$ is a (discrete) probability distribution on a set of m elements. Let $N_{a_j}(\mathbf{X})$ denote the number of times the observation $\mathbf{X}$ takes on the value a_j. Then obviously $\sum_i N_{a_j}(\mathbf{X}) = n$. The χ^2 statistic is the random variable defined by

$$\chi^2 = \sum_{j=1}^{m} (N_{a_j}(\mathbf{X}) - np(j))^2 / np(j) \tag{1}$$

For the uniform distribution p_U, the χ^2 statistic is just $m/n \sum_i (N_{a_j}(\mathbf{X}) - n/m)^2$. In a χ^2 test, the observed χ^2 statistic is compared to $\chi^2_{a,m-1}$, the threshold for the χ^2 test with $m-1$ degrees of freedom and with significance level a. In our investigation of RC6, we shall specifically need the threshold values for 1023 degrees of freedom, as shown in Tables 1 and 2. For example, the entry 1131 for 0.99 in Table 1 says that the expression $m/n \sum_i (N_{a_j}(\mathbf{X}) - n/m)^2$ for large n will exceed 1131 only in 1% of the time, provided the underlying distribution of the observation $\mathbf{X}$ is indeed uniform.
For practical experiments the question arises how large the size n of the observation should be in order to detect that a distribution p_X is nonuniform. In order to estimate n, consider the bias of a probability distribution p_X defined by the

Table 1. Selected threshold values of the χ^2 distribution with 1023 degrees of freedom.

Level	0.50	0.60	0.70	0.80	0.90	0.95	0.99	0.999	0.9999
χ^2	1022	1033	1046	1060	1081	1098	1131	1168	1200

Table 2. Selected threshold values of the χ^2 distribution with 1023 degrees of freedom.

Level	$1-2^{-16}$	$1-2^{-24}$	$1-2^{-32}$	$1-2^{-48}$	$1-2^{-64}$
χ^2	1222	1280	1330	1414	1474

distance measure

$$||p_X - p_U|| = \sum_j (p_X(j) - p_U(j))^2 \tag{2}$$

From [7] we quote the expected value of the χ^2 statistic (1) of a distribution p_X, as well as some useful conclusions:

$$\mathbf{E}_X\chi^2 = nm||p_X - p_U|| + m - m||p_X|| \tag{3}$$

For the case of the uniform distribution this implies $\mathbf{E}_U\chi^2 = m-1$. Moreover it follows that for $n = c/||p_X - p_U||$ the expected value is $\mathbf{E}_X\chi^2 = cm + m - m||p_X||$. Since in practical cases often $||p_X|| \approx ||p_U||$, this simplifies to $\mathbf{E}_X\chi^2 \approx (c+1)m - 1$. Thus $\mathbf{E}_X\chi^2$ differs from $\mathbf{E}_U\chi^2$ significantly, if $c = \Omega(1)$. As a conclusion, the size $n = c/||p_X - p_U||$ of the observation suffices to distinguish a source with distribution p_X from a source with uniform distribution. Clearly, the constant c needs to be larger for higher significance level a.

3 Correlations in RC6

In [4], under the title of Type I Approximations, a two-round linear approximation has been studied which is based on small rotation amounts in the data dependent rotations. This linear approximation is described by $(A \cdot e_t) \oplus (C \cdot e_s) = (A'' \cdot e_u) \oplus (C'' \cdot e_v)$. Here A and C are the first and third words of some intermediate data, A'' and C'' are the first and third words of the intermediate data after a further two rounds of encryption in RC6, and e_t denotes the 32-bit word with a single one in the t^{th} least significant bit position. It has been noticed that for $t = s = u = v = 0$ the case where both rotation amounts are zero in the first of the two rounds leads to a bias of 2^{-11}. This is derived by using the piling-up lemma and the fact that the second and fourth words remain unchanged in the second round. If t, s, u, v are nonzero but less than 5, there is a smaller bias, which depends on the values of t, s, u, v. Note that no key bits are involved in the approximation.

In our approach we do not consider the XOR of single bits in the first and third words. Instead we fix each of the least significant five bits in words A and C of the input and investigate the statistics of the 10-bit integer obtained by concatenating each of the least significant five bits in words A'' and C'' every two rounds later. This is motivated by the fact that the least significant five bits in A and C altogether are not changed by the xor and data dependent rotation if both rotation amounts are zero. More generally, we can expect a bias for amounts smaller than five. As we shall demonstrate, this leads to much stronger biases which can be iterated over many rounds, just as linear approximations. In this way we can consider small rotation amounts as a single event, in which amounts near zero from the negative, like 30 or 31, prove to be useful as well.

3.1 Small Rotation Amounts

To see the effect of small rotation amounts on the values of the least significant five bits in the first and third words in RC6, we implemented the following tests with 4 rounds:

Let us denote by (a, b) the two amounts in the data dependent rotations in the first round. To measure the effect on the distribution of the target bits, we forced the values of a and b by taking appropriate plaintexts and we computed the χ^2-value of the 10-bit integers after 4 rounds. For each experiment we took 2^{18} texts to get a big χ^2-value to clearly measure the effect.

Table 3. Statistical effect of small rotation amounts

a, b	χ^2
0,0	2775
0,31	2107
0,1	1998
31,31	1715
1,1	1643
0,30	1633
0,2	1572
30,31	1388
1,2	1326
0,3	1306
0,4	1145
0,5	1053

By the symmetry in the design of RC6, it can be expected that (a, b) gives the same χ^2-value as (b, a).

We observe that the χ^2-values for all pairs (a, b) with $|a| < 5$ and $|b| < 5$ are significantly higher than the expected value 1023 for uniform 10-bit integers. Note that these tests suggest that we get similar χ^2-values for constant values of the "distance" $|a-32|+|b-32| \pmod{32}$: The pairs (0,31) and (0,1) both have distance 1 and similar χ^2-values, and the pairs (1,1),(0,2), (31,31), and (0,30) all have distance 2 and have similar χ^2-values.

Let us take a closer look at how the above observations lead to a nonuniform distribution. Assume that the least significant five bits of plaintext words A and C are fixed, e.g., to zero bits. Let us denote by X the concatenation of the least significant five bits of the ciphertext words A and C after two rounds of encryption. In this example, for illustration, we will ignore the addition of the subkeys in the output transformation, and also we will assume that the least significant five bits of both round keys $S[2]$ and $S[3]$ are zero. Denote by t_5 and u_5 the least significant five bits of t and u, see Figure 1. Then in the first round, if $t_5 = u_5 = 0$, then X will be zero. Since the function f is a permutation, t_5 and u_5 will be zero with a probability of 2^{-5} each. If we assume that for $|t_5| \geq 5$ and $|u_5| \geq 5$, the values of X are distributed uniformly at random, the probability that X is zero is at least $2^{-10}+(23/32 \cdot 1/32)^2 \simeq 2^{-10}+2^{-10.95}$. With rotations $t_5 = 1, u_5 = 0$, X will take the possible values (in bits) 0000b00001, where 'b' is a random bit. With rotations $t_5 = 0, u_5 = 1$, X will take the possible values (in bits) 000010000b. Thus, X = 0000100001 with probabilty at least $2 \cdot 2^{-11} + (23/32 \cdot 1/32)^2$. Note that both these estimates are lower bounds. E.g., in the case where $t_5 = u_5 = 4$, X will take the possible values (in bits) $0b_1b_2b_3b_40b_5b_6b_7b_8$, and in the case where $t_5 = 1, u_5 = 16$, X will take the possible values (in bits) $b_1b_2b_3b_4b_50000b_6$, where the b_is are random bits. Thus, X can take both the values 0000000000 and 0000100001 also in these cases.

It has been clearly demonstrated that the distribution of X is nonuniform. Note that although it was assumed that the involved subkey bits were zero, it follows easily that the nonuniformity remains when these key bits are randomly chosen.

3.2 χ^2 Statistic of RC6

Here, we investigate the nonrandomness of r-round versions of RC6. This analysis is based on systematic experiments on increasing numbers of rounds of RC6 with varying word length w. Our method is used to demonstrate that detecting and quantifying nonrandomness is experimentally feasible up to 6 rounds of RC6.

For this purpose, the least significant lg w bits in words A and C of the input are fixed to zero. Depending on the experiment and the number of rounds, the remaining input bits are either chosen randomly, or more of the remaining input bits are suitably fixed so that one (or both) of the data dependent rotations are zero. In our tests, we persue the χ^2 statistic of the integer of size twice lg w bits as obtained by concatenating the least significant lg w bits in words A'' and C'' every two rounds later.

In the experiments, we consider versions of RC6 with word size $w = 8, 16$ and 32 bits, respectively ($w = 32$ corresponding to the AES candidate RC6). It is instructive to see that the general behaviour of the χ^2 test for increasing numbers of rounds in all three cases is very similar. To judge the outcome of these χ^2 tests note that for the word sizes w as considered, 6-bit, 8-bit and 10-bit integers are tested at the output. Hence the numbers of freedom are 63, 255 and 1023 respectively, and these numbers coincide with the expected value of the χ^2 statistic, provided the distribution to be tested is uniform.

Subsequently we discuss the results of implemented tests in more detail, where the keys are chosen at random.

32-bit RC6. First consider a version of RC6 with block length of 32 bit. This corresponds to the case $w = 8$, which is shown in Table 4. For $r = 2$ and $r = 4$ rounds more than one entry is given. The first entry shows a number of texts, measured in powers of two, which is necessary to detect that the mean of the χ^2 values over 20 tests is higher than the expected value 63 if the distribution would be random. The other entries show a significant increase of this mean if the number of plaintexts is doubled, thus a strong deviation from the uniform distribution. For 2^8, 2^{17} and 2^{26} texts and correspondingly for 2, 4 and 6 rounds, the χ^2 values are approximately the same. Thus we have to increase the number of plaintexts by the same factor 2^9 for every two more rounds to get a comparable statistical deviation as measured by the χ^2 test. For this small version of RC6 we cannot go beyond 6 rounds, as we have to fix 6 input bits, and for 6 rounds we already need 2^{26} random texts.

Table 4. RC6 with 32-bit blocks and r rounds. Expected χ^2 for a random function is 63.

r	#Texts	χ^2	#Tests
2	2^8	77	20
2	2^9	107	20
4	2^{16}	68	20
4	2^{17}	73	20
4	2^{18}	83	20
6	2^{26}	78	20

64-bit RC6. Next consider the version of RC6 with word size $w = 16$, i.e. RC6 with 64-bit blocks. The results are shown in Table 5. Here the expected value of the χ^2 statistic is 255. Again a substantial increase is observed in the mean for χ^2-values if the number of texts is doubled. We notice that passing from 2 to 4 to 6 rounds, the averaged χ^2-values increase slightly if the corresponding number of plaintexts is increased by a constant factor of 2^{13}.

128-bit RC6. Consider now r-round versions of RC6 with word size 32 bits, i.e. with round function as in the AES proposal. Table 6 shows the results of implemented tests for $r = 2$ and $r = 4$ rounds. Recall that for 10-bit integers the expected value of the χ^2 statistic is 1023, and according to Table 1 the 95% significance level is 1098 and the 99% significance level is 1131. Thus all tests as reported in Table 6 are very unlikely to be produced by uniformly distributed

Table 5. RC6 with 64-bit blocks and r rounds. Expected χ^2 for a random function is 255.

r	#Texts	χ^2	#Tests
2	2^{10}	283	100
2	2^{11}	308	100
2	2^{12}	364	100
4	2^{23}	286	100
4	2^{24}	318	100
6	2^{36}	298	10

10-bit integers. In fact for 4 rounds and 2^{33} texts almost twice the expected value for a uniform distribution is achieved.

Table 6. RC6 with 128-bit blocks and r rounds. Expected χ^2 for a random function is 1023.

r	#Texts	χ^2	#Tests
2	2^{13}	1096	20
2	2^{14}	1196	20
2	2^{15}	1332	20
2	2^{16}	1649	20
2	2^{17}	2208	20
4	2^{29}	1096	20
4	2^{30}	1163	20
4	2^{31}	1314	20
4	2^{32}	1527	20
4	2^{33}	2054	20

Table 7 shows the results of tests with up to 6 rounds but with one or both data dependent rotations in the first round to be fixed to zero. The last entry is the result of a test run on eight processors of a Cray Origin 2000 computer. Both, the experiments in Table 6 and in Table 7 demonstrate that for up to 6 rounds each additional two rounds require roughly 2^{16} times as many texts to get about the same χ^2-value on average. The first two entries of Table 7 again show an increase of the χ^2-values if the number of texts is doubled.

Table 7. RC6 with 128-bit blocks and r rounds. Expected χ^2 for a random function is 1023.

r	#Texts	χ^2	#Tests	Comments
4	2^{22}	1124	20	zero rotation in 1. round at word D
4	2^{23}	1228	20	zero rotation in 1. round at word D
6	2^{38}	1106	1	zero rotation in 1. round at word D

3.3 A Possible Analytical Explanation

In this subsection we make an attempt to analytically predict the complexities of the χ^2 tests on RC6.

In the following, let X be the random variable representing the 10 bits as considered in the ciphertexts after 2 rounds of encryption with RC6 in the tests from the preceding section. Also, let Y and Z be the random variables representing these 10 bits in the ciphertexts after 4 respectively 6 rounds of encryption. It follows from the description of RC6, that the 10 bits in the ciphertexts after six rounds are not the exclusive-or of 10 biased bits from the first two rounds and 10 biased bits from the next two rounds. This is due to the fact that the data-dependent rotations in RC6 are performed after the exclusive-or with the data from the previous rounds. Thus, a parallel to the Piling-Up Lemma used by Matsui [11] does not seem to be applicable.

With the test results of the preceding section and the estimate from Sec. 2, that with $n = c/||p_X - p_U||$ texts one can expect a χ^2-value of $(c+1)m$, it is possible to compute estimates of $||p_X - p_U||$, $||p_Y - p_U||$, and $||p_Z - p_U||$.

64-bit RC6. The results of the tests in Table 5 yield the following estimates for the distances:
$||p_X - p_U|| = 2^{-13.25}$, $||p_Y - p_U|| = 2^{-26.03}$, $||p_Z - p_U|| = 2^{-38.57}$. Thus, this is a clear indication that $||p_Y - p_U|| > ||p_X - p_U||^2$, and that $||p_Z - p_U|| > ||p_X - p_U|| \cdot ||p_Y - p_U||$.

This gives perhaps more convincing evidence, that passing from s to $s+2$ rounds in the tests of the preceding section, requires an increase in the texts needed of a factor of a little less than 2^{13}.

128-bit RC6. The results of the tests in Table 6 with a χ^2-value greater than 1300 yield the following estimates for the distances:
$2^{-16.79} \leq ||p_X - p_U|| \leq 2^{-16.71}$, $2^{-33.02} \leq ||p_Y - p_U|| \leq 2^{-32.81}$. Again with a clear indication that $||p_Y - p_U|| > ||p_X - p_U||^2$.

This confirms the estimate from the preceding section that passing from s to $s+2$ rounds in the χ^2-tests, requires an increase in the texts needed of a factor of a little more than 2^{16}. Later, we will use the factor $2^{16.2}$.

3.4 Weak Keys

The test results from the previous sections were given as an average over tests using randomly chosen keys. There was some deviation of the single results, e.g., the χ^2-values of the tests for RC6 with 128-bit blocks and 4 rounds using 2^{33} texts varied from 1731 to 2595 with an average of 2044. Thus, for some keys the deviation is bigger than expected, for other keys it is lower than expected. In this section we report on some weak keys, which perform better than the average key. In Sec. 3.1 it was explained why there is a nonuniform distribution of the 10 target bits, and why for two rounds the involved key bits have no influence on the nonuniformity of the target bits in the χ^2-tests. However, when iterating the tests to several rounds, the modular additions of round-key bits introduce carry bits which affect the nonuniformity. For 4 rounds, the key bits that may affect the nonuniformity are the five least significant bits of the round keys $S[2]$ and $S[3]$. When these bits are set to zeros, the χ^2-value increases. Similarly, for 6 rounds the least significant five bits of the subkeys 2,3,6, and 7 may influence the nonuniformity.

This is illustrated by a series of tests, the results of which are shown in Table 8.

Table 8. RC6 with 128-bit blocks and r rounds for weak keys.

r	#Texts	χ^2	#Tests	Comments
4	2^{30}	1398	20	1 in 2^{10} keys
6	2^{30}	1093	10	zero rotation in 1.round at B and D
6	2^{30}	1368	10	same, for 1 in 2^{20} keys

For 4 rounds the "distance" to a uniform distribution is about $2^{-31.5}$ which is more than a factor of two higher than for the results averaged over all keys. For 6 rounds the distance to the uniform distribution is about $2^{-33.87}$ for the second test of Table 8, and about $2^{-31.57}$ for the third test using weak keys. Thus, a factor of more than 4.

4 Attacks on RC6

4.1 Distinguishing Attacks

It is possible to exploit the findings in the previous sections to distinguish RC6 with a certain number of rounds from a permutation randomly chosen from the set of all permutations. In the previous sections we fixed bits in the first and third plaintext words. As we shall see in the next section this makes good sense when implementing key-recovery attacks. In a distinguishing attack it is advantageous to fix the least significant five bits in the second and fourth words instead. It follows that after one round of encryption the least significant five bits in the first and third words of the ciphertext are constant. Table 9 lists the result of tests implemented for RC6 with 128-bit blocks with 3 and 5 rounds. It follows that $2^{13.8}$ texts are sufficient to distinguish the 3-round encryption permutation from a randomly chosen permutation in 90% of the cases. We estimate that for RC6 with $3 + 2r$ rounds similar results will hold using $2^{13.8+r\times16.2}$ texts, which is confirmed by tests implemented on RC6 with 5 rounds.

Note that the χ^2 numbers of Table 9 for 3 rounds are slightly lower than the numbers of Table 6 for 2 rounds. This stems from the fact that in the latter tests, the least significant five bits of the first and third words of the plaintexts were fixed to zeros. In a distinguising attack, one gets the first round "for free", by fixing totally 10 bits of the second and fourth words. However, as these words are added modular 2^{32} to subkeys in the input transformation, the least significant five bits of the first and third words in the inputs to the second round are nonzero, but constant, and there is an effect of carry bits by the addition of subkeys after the second-round approximation.

We estimate that for keys where the least significant five bits of each of the two subkeys in every second round are zeros, the attack improves with more than a factor of two for each 2 rounds. This leads to the estimate that for one in 2^{80} keys, 17 rounds of RC6 with 128-bit blocks can be distinguished from a randomly chosen permutation.

Table 9. Complexities for distinguishing RC6 with 128-bit blocks and r rounds from a random function.

r	#Texts	χ^2	Comments
3	2^{13}	1079	Implemented, average 20 tests
3	$2^{13.8}$	1100	Implemented, average 20 tests
3	2^{14}	1141	Implemented, average 20 tests
5	2^{29}	1054	Implemented, average 20 tests
5	2^{30}	1099	Implemented, average 20 tests
7	$2^{46.2}$		Estimated.
9	$2^{62.4}$		Estimated.
11	$2^{78.6}$		Estimated.
13	$2^{94.8}$		Estimated.
15	$2^{111.0}$		Estimated.
17	$\leq 2^{118}$		Estimated. For 1 in every 2^{80} keys.

4.2 Key-Recovery

As confirmed by several experiments, the χ^2-value is significantly higher if inputs are suitably fixed so that one (or both) of the data dependent rotations in the first round of RC6 are zero. Clearly, the choice of the right input depends on knowledge of the subkey $S[0]$ (or $S[1]$, respecively). We now describe how the considerations and experimental results of previous sections can be exploited for key recovery. Thereby we restrict to 128-bit RC6 with word size 32 bits.

In the following we will assume that to get similar values in a χ^2-test on $s+2$ rounds compared to s rounds requires a factor of $2^{16.2}$ additional plaintexts. Recall that we always fix the least significant five bits in words A and C. In addition suppose we fix inputs so that the data dependent rotation is zero in the first round at word D. Then with a factor of about $2^{8.1}$ less plaintexts we achieve a similar χ^2-value as for random inputs at word D (e.g. compare row 7 in Table 6 with the first row in Table 7). For symmetry reasons, the same holds if inputs at word B are fixed.

With regard to inputs at word D (or B), some comments related to the multiplication in RC6 are in order. The data dependent rotation amounts are determined by the five leading bits of the output of the permutation as given by the multiplication $D \times (2D+1)$ (see Figure 1). The permutation function restricted to these five output bits is therefore balanced. Rather than fixing inputs we can restrict to inputs leading to these five bits being zero, resulting in more freedom for choosing plaintexts. For efficiency we can prepare a table T of the 2^{27} inputs to the permutation giving zero rotation. Thus for the correct key $S[1]$ we can choose 2^{27} different inputs at word D, all leading to zero rotation in the right half of the first round. (Alternatively, we can enlarge the table, and also accept inputs giving rotation amount 1 or -1, which still lead to increased χ^2-values.) To test a fixed trial key $S[1]$ we thus can roughly choose amongst 2^{113} plaintexts at random.

The attack goes as follows, choose plaintexts such that the least significant five bits of the first and third words are zeros. Prepare an array with 2^{10} entries for each value of the subkey $S[1]$. For each plaintext use the table T as prepared, to determine the values of $S[1]$ which lead to a zero rotation at word D. For each such value, update each array by incrementing the entry corresponding to the value obtained from the 10 target bits of the ciphertext. Each array is used to find the probability distribution of the 10 target bits. Repeat the attack sufficiently many times, until one array has a significantly higher value in the χ^2-test.

For an estimate of the complexity to recover subkey $S[1]$, consider r-round versions of RC6 with r even. For each trial key $S[1]$ we perform a χ^2 test with

$$2^{13} \times (2^{16.2})^{\frac{r-2}{2}} \times 2^{-8.1} \tag{4}$$

plaintexts as described. Then for the correct choice of $S[1]$ the χ^2-value is expected to be around 1100, that is, significantly higher than 1023. For each key which produces an expected χ^2-value, repeat the attack with additional plaintexts.

To rule out all false values of the key, we increase the number of texts by up to a factor of 2^3. Enlarging the amount of plaintexts by this factor has the effect of a substantial increase of the χ^2-value, as observed in our experiments (see the tables in Section 3). Thus, to single out the correct key out of suggested key values we would need about $2^{16} \times (2^{16.2})^{\frac{r-2}{2}} \times 2^{-8.1}$ texts. And since only one in every 2^5 texts gives the desired zero rotation at word D, the total number of plaintexts needed is

$$2^5 \times 2^{16} \times (2^{16.2})^{\frac{r-2}{2}} \times 2^{-8.1} = 2^{r\times 8.1-3.3}.$$

The amount of work is estimated as follows. For each plaintext in the attack, we update the counters of at most 2^{27} keys. If we assume that after the first two iterations of the attack, the number of remaining keys are reduced by a factor of 4 or more, we obtain a complexity of

$$2^{27+r\times 8.1-5.3} = 2^{21.7+r\times 8.1},$$

where one unit is the time to update one entry of one array of size 2^{10} of totally 2^{32} arrays.

After $S[1]$ is correctly found, subkey $S[0]$ can be determined with a reduced amount of texts and work. Knowing $S[0]$ and $S[1]$, the data dependent rotations in the first round can be fixed to zero without effort. Thus the χ^2 tests can now be applied by controlling inputs to the second round. This enables finding subkeys $S[2]$ and $S[3]$ in much the same way as we did for $S[0]$ and $S[1]$. After this we peel of the first round and proceed to determine the other subkeys.

This attack is faster than an exhaustive key search for the 128-bit key version of RC6 with up to 12 rounds, and for the 192-bit and 256-bit versions of RC6 with up to 14 rounds. Table 10 lists the complexity for 12, and 14 rounds of RC6. For 16 rounds the number of texts needed is $2^{126.3}$ and thus exceeds the number of available texts of 2^{118}.

For key sizes 192 bits and 256 bits the computational effort for searching subkeys can be larger. Thus for a 192-bit key we can do a simultaneous search over $S[0]$ and $S[1]$, thereby improving the χ^2 statistic by two rounds. In addition, we increase the factor 2^3 to 2^4 in order to single out the correct pair $S[0]$, $S[1]$ among the remaining pairs. Here only one in every 2^{10} plaintexts give zero rotations at words B and D. The number of plaintexts needed for this version of the attack is

$$2^{10} \times 2^{17} \times (2^{16.2})^{\frac{r-2}{2}-1} = 2^{r\times 8.1-5.4},$$

and the time complexity is

$$2^{54+r\times 8.1-7.4} = 2^{46.6+r\times 8.1},$$

where one unit is the time to update one entry of one array of size 2^{10} of totally 2^{64} arrays. Table 10 lists the complexities of this attack for 14 rounds of RC6. The number of texts needed in the attack on 16 rounds is about 2^{124} and thus still exceeds 2^{118}. However, as reported earlier there are keys for which the complexities improve. We estimate that the attack is possible for at least one in 2^{60} keys with the complexity as stated in the table.

Finally, for the 256-bit key version of RC6 it is possible to further extend the attack. In a 15-round version, one can search over the keys $S[0], S[1], S[32]$, and $S[33]$. The latter two keys are used to decrypt the ciphertexts one round. In the updating of the probability-arrays, one only uses ciphertexts for which there are

Table 10. Complexities for key-recovery attacks on RC6 with 128-bit blocks and r rounds. One unit in "Work" is the time to increment one counter. The χ^2 value is the expected value for the correct key.

r	#Texts	Work	χ^2	Memory	Comment
12	2^{94}	2^{119}	1500	2^{42}	
14	2^{110}	2^{135}	1500	2^{42}	
14	2^{108}	2^{160}	2000	2^{74}	
16	2^{118}	2^{171}	2000	2^{74}	1 in 2^{60} keys
15	2^{119}	2^{215}	> 2000	2^{138}	

zero rotations in the last round. The number of texts needed is approximately 2^{10} times that of 14 rounds, and the time complexity increases with a factor of about 2^{54}. To rule out all false values of the keys, we estimate that the number of plaintexts needed increases by yet a factor of 2.

Note that in the above attacks the number of available texts is bounded by 2^{118}, since we need to fix 10 bits of each plaintext. The probability distributions for each such fixed 10-bit value will be different, but their distance to the uniform distribution can be expected to be similar. As an extension of the above attacks consider the following. Run the attack with x texts for one fixed value of the 10 bits in the plaintexts. Record the χ^2-value for each key in the attack, and rank the keys. Reset the arrays. Repeat the attack x texts for another fixed value of the 10 bits. Record again the χ^2-value for each key in the attack, and rank the keys. Repeat this a number of times. If the χ^2-values for the correctly guessed keys will be larger than for random values, one can expect that the correct key will be high in the rankings, and it can be detected after sufficiently many iterations. Thus this variant would make available all 2^{128} texts. We conjecture that this attack is applicable to 15 rounds of RC6 with a complexity as given in the last entry of Table 10.

We leave it as an open question whether the attack and its variants can be used to attack RC6 with 16 or more rounds.

Finally, note that the reported attacks are chosen plaintext attacks. However, it follows that the basic attack reported earlier can be easily transformed into a known plaintext attack with an increase in the needed texts of a factor of at most 2^{10}, leaving the total time complexity unaltered.

5 Conclusion

In this paper we have presented an attack on RC6 which is based on a strong relationship between the effects of data dependent rotations in the round function and statistical input-output dependencies.

Estimates which are based on systematic experimental results show that versions of RC6 with up to 15 rounds can be distinguished from a random permutation. A class of weak keys has been identified for which this nonrandomness is estimated to persist up to 17 rounds. Finally, we have derived a method for key-recovery for RC6 with up to 15 rounds which is faster than exhautive key search. We do not know whether our analysis can be used to attack RC6 with 16 or more rounds.

We remark that similar attacks are applicable to reduced-round versions of RC5. However, it seems such attacks are not better than existing (differential) attacks on RC5 [3].

Acknowledgments. The authors would like to thank Vincent Rijmen for helpful comments and discussions.

References

1. O. Baudron, H. Gilbert, L. Granboulan, H. Handschuh, A. Joux, P. Nguyen, F. Noilhan, D. Pointcheval, T. Pornin, G. Poupard, J. Stern, and S. Vaudenay Report on the AES candidates. Available at `http://csrc.nist.gov/encryption/aes/round1/conf2/papers/baudron1.pdf`.
2. H. Gilbert, H. Handschuh, A. Joux, and S. Vaudenay. A Statistical Attack on RC6. These proceedings.
3. A. Biryukov and E. Kushilevitz. Improved cryptanalysis of RC5. In K. Nyberg, editor, *Advances in Cryptology - EUROCRYPT'98, LNCS 1403*, pages 85–99. Springer Verlag, 1998.
4. S. Contini, R.L. Rivest, M.J.B. Robshaw and Y.L. Yin. The Security of the RC6 Block Cipher. v.1.0, August 20, 1998. Available at `www.rsa.com/rsalabs/aes/`.
5. S. Contini, R.L. Rivest, M.J.B. Robshaw, and Y.L. Yin. Improved analysis of some simplified variants of RC6. In L. Knudsen, editor, *Fast Software Encryption, Sixth International Workshop, Rome, Italy, March 1999, LNCS 1636*, pages 1–15. Springer Verlag, 1999.
6. S. Contini, R.L. Rivest, M.J.B. Robshaw, and Y.L. Yin. Some Comments on the First Round AES Evaluation of RC6. Available at `http://csrc.nist.gov/encryption/aes/round1/pubcmnts.htm`.
7. J. Kelsey, B. Schneier, and D. Wagner. Mod n cryptanalysis, with applications against RC5P and M6. In L. Knudsen, editor, *Fast Software Encryption, Sixth International Workshop, Rome, Italy, March 1999, LNCS 1636*, pages 139–155. Springer Verlag, 1999.
8. A.G. Konheim. Cryptography: A Primer. John Wiley & Sons, 1981.
9. L.R. Knudsen, and W. Meier. Correlations in RC6. Technical Report 177, Department of Informatics,University of Bergen, Norway, July 29, 1999.
10. D.E. Knuth. The Art of Computer Programming, Vol. 2. Addison-Wesley, 1981.
11. M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor, *Advances in Cryptology - EUROCRYPT'93, LNCS 765*, pages 386–397. Springer Verlag, 1993.
12. R.L. Rivest, M.J.B. Robshaw, R. Sidney and Y.L. Yin. The RC6 Block Cipher. v1.1, August 20, 1998. Available at `www.rsa.com/rsalabs/aes/`.
13. S. Vaudenay. An Experiment on DES Statistical Cryptanalysis. 3rd ACM Conference on Computer and Communications Security, ACM Press, 1996, pp. 139-147.

On the Interpolation Attacks on Block Ciphers

A.M. Youssef and G. Gong

Center for Applied Cryptographic Research
Department of Combinatorics and Optimization
University of Waterloo, Waterloo, ON N2L 3G1
{a2youssef, ggong}@cacr.math.uwaterloo.ca

Abstract. The complexity of interpolation attacks on block ciphers depends on the degree of the polynomial approximation and/or on the number of terms in the polynomial approximation expression. In some situations, the round function or the S-boxes of the block cipher are expressed explicitly in terms of algebraic function, yet in many other occasions the S-boxes are expressed in terms of their Boolean function representation. In this case, the cryptanalyst has to evaluate the algebraic description of the S-boxes or the round function using the Lagrange interpolation formula. A natural question is what is the effect of the choice of the irreducible polynomial used to construct the finite field on the degree of the resulting polynomial. Another question is whether or not there exists a simple linear transformation on the input or output bits of the S-boxes (or the round function) such that the resulting polynomial has a less degree or smaller number of non-zero coefficients. In this paper we give an answer to these questions. We also present an explicit relation between the Lagrange interpolation formula and the Galois Field Fourier Transform.

Keywords: Block cipher, cryptanalysis, interpolation attack, finite fields, Galois Field Fourier Transform

1 Introduction

Gong and Golomb [7] introduced a new criterion for the S-box design. Because many block ciphers can be viewed as a Non Linear Feedback Shift Register (NLFSR) with input then the S-boxes should not be approximated by a monomial. The reason is that the trace functions $Tr(\zeta_j X^d)$ and $Tr(\lambda X)$ have the same linear span. From the view point of m-sequences [10], both of the sequences $\{Tr(\zeta\alpha^{id})\}_{i\geq 0}$ and $\{Tr(\lambda\alpha^i)\}_{i\geq 0}$ are m-sequences of period $2^n - 1$. The former can be obtained from the later by decimation d. Gong and Golomb showed that the distance of DES S-boxes approximated by monomial functions has the same distribution as for the S-boxes approximated by linear functions.

In [3] Jakobsen and Knudsen introduced a new attack on block ciphers. This attack is useful for attacking ciphers using simple algebraic functions as S-boxes. The attack is based on the well known Lagrange interpolation formula. Let R be a field. Given $2n$ elements $x_1, \ldots, x_n, y_1, \ldots, y_n \in R$, where the x_is are distinct. Define

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

Then $f(x)$ is the only polynomial over R of degree at most $n - 1$ such that $f(x_i) = y_i$ for $i = 1, \ldots, n$. The main result in [3] is that for an iterated block cipher with block size m, if the cipher-text is expressed as a polynomial with $n \leq 2^m$ coefficients of the plain-text, then there exists an interpolation attack of time complexity n requiring n known plain-texts encrypted with a secret key K, which finds an algorithm equivalent to encryption (or decryption) with K. This attack can also be extended to a key recovery attack.

In [4] Jakobsen extended this cryptanalysis method to attack block ciphers with probabilistic nonlinear relation of low degree. Using recent results from coding theory (Sudan's algorithm for decoding Reed-Solomon codes beyond the error correction parameter[6]), Jakobsen showed how to break ciphers where the cipher-text is expressible as evaluations of unknown univariate polynomial of low degree m with a typically low probability μ. The known plain-text attack requires $n = 2m/\mu^2$ plain-text/cipher-text pairs. In the same paper, Jakobsen also presented a second attack that needs access to $n = (2m/\mu)^2$ plain-text/cipher-text pairs and its running time is polynomial in n.

It is clear that the complexity of such cryptanalytic attacks depends on the degree of the polynomial approximation or on the number of terms in the polynomial approximation expression. In some situations, the round function or the S-boxes of the block cipher are expressed explicitly in terms of algebraic function (For example see [8]), yet in many other occasions the S-boxes are expressed in terms of their Boolean function representation. In this case, the cryptanalyst has to evaluate the algebraic description of the S-boxes or the round function using the Lagrange interpolation formula. A natural question is what is the effect of the choice of the irreducible polynomial used to construct the finite field on the degree of the resulting polynomial. Another question is whether or not there exists a simple linear transformation on the input or output bits of the S-boxes (or the round function) such that the resulting polynomial has a less degree or smaller number of coefficients. In this paper we give explicit answer to these questions. To illustrate the idea, consider the binary mapping from $GF(2)^4$ to $GF(2)^4$ given in the Table 1. If the Lagrange interpolation formula is applied to $GF(2^4)$ where $GF(2^4)$ is defined by the irreducible polynomial $X^4 + X^3 + 1$ then we have $F(X) = X + X^2 + 7X^3 + 15X^4 + 5X^5 + 14X^6 + 14X^8 + 2X^9 + 7X^{10} + 9X^{12}, X \in GF(2^4)$. However, if we use the irreducible polynomial $X^4 + X + 1$ to define $GF(2^4)$ then we have $F(X) = X^3, X \in GF(2^4)$ which is obviously a simpler description.

An interesting observation follows when applying the Lagrange interpolation formula to the DES S-boxes. In this case we consider the DES S-boxes output

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f(x)$	0	1	8	15	12	10	1	1	10	15	15	12	8	10	8	12

Table 1.

coordinates as a mapping from $GF(2^6)$ to $GF(2)$. Let f be the Boolean function resulting from XORing all the output coordinates of the DES S-box number four. When we define $GF(2^6)$ using the irreducible polynomial $X^6 + X^5 + 1$, the polynomial resulting from applying the Lagrange interpolation formula to f has only 39 nonzero coefficient. The Hamming weight of all the exponents corresponding to the nonzero coefficients was ≤ 3. It should be noted that the expected value of the number of nonzero coefficients for a randomly selected function over $GF(2^6)$ is 63. While this observation doesn't have a cryptanalytic significance, it shows the effect of changing the irreducible polynomial when trying to search for a polynomial representation for cipher functions.

2 Mathematical background and definitions

For a background about the general theory of finite fields, the reader is referred to [1] and for a background about finite fields of charachteristic 2, the reader is referred to [2].

Most of the results in this paper can be extended in a straightforward way from $GF(2^n)$ to $GF(q^n)$. Throughout this paper, we use integer labels to present finite field elements. I.e., for any element $X \in GF(2^4)$, $X = \sum_{i=0}^{n-1} x_{i+1}\alpha^i$, $x_i \in GF(2)$ where α is a root of the irreducible polynomial which defines $GF(2^n)$, we represent X by $\sum_{i=0}^{n-1} x_{i+1}2^i$ as an integer in the range $[0, 2^n - 1]$. The associated addition and multiplication operations of these labels are defined by the finite field structure and have no resemblance to modular integer arithmetic.

Definition 1. A polynomial having the special form

$$L(X) = \sum_{i=0}^{t} \beta_i X^{2^i} \tag{2}$$

with coefficients β_i from $GF(2^n)$ is called a linearized polynomial over $GF(2^n)$.

Definition 2. A cyclotomic coset mod N that contains an integer s is the set

$$C_s = \{s, sq, \ldots, sq^{m-1}\} \pmod{N} \tag{3}$$

where m is the smallest positive integer such that $sq^m \equiv s \pmod{N}$.

Lemma 3. *Let A be a linear mapping over $GF(2^n)$, then $A(X), X \in GF(2^n)$ can be expressed in terms of a linearized polynomial over $GF(2^n)$. I.e., we can express $A(X)$ as*

$$A(X) = \sum_{i=0}^{n-1} \beta_i X^{2^i} \tag{4}$$

Lemma 4. *Let $\alpha_1, \alpha_2, \ldots, \alpha_t$ be elements in $GF(2^n)$. Then*

$$(\alpha_1 + \alpha_2 + \ldots + \alpha_t)^{2^k} = \alpha_1^{2^k} + \alpha_2^{2^k} + \ldots + \alpha_t^{2^k} \tag{5}$$

Lemma 5. *The number of ways of choosing a basis of $GF(2^n)$ over $GF(2)$ is*

$$\prod_{i=0}^{n-1} (2^n - 2^i) \tag{6}$$

3 Lagrange coefficients, Galois Field Fourier Transform and Boolean functions

3.1 Relation between the Galois Field Fourier Transform and the Lagrange coefficients

In this section we give an explicit formula for the relation between the Lagrange Interpolation of F and the Galois Field Fourier Transform of its corresponding sequence. Besides its theoretical interest, the cryptographic significance of this relation stems from the view point of Gong and Golomb [7] where they model many block ciphers as a Non Linear Feedback Shift Register (NLFSR) with input.

Let $\mathbf{v} = (v_0, v_1, \ldots, v_{l-1})$ be a vector over $GF(q)$ whose length l divides $q^m - 1$ for some integer positive m. Let α be an element of order l in $GF(q^m)$. The Galois field Fourier transform (GFFT) [11] of $\mathbf{v}$ is the vector $\mathcal{F}(\mathbf{v}) = \mathbf{V}$ $= (V_0, V_1, \ldots, V_{l-1})$ where $\{V_j\}$ are computed as follows.

$$V_j = \sum_{i=0}^{l-1} \alpha^{-ij} v_i, j = 0, 1, \ldots, l-1. \tag{7}$$

The inverse transform is given by

$$v_i = \frac{1}{l} \sum_{j=0}^{l-1} \alpha^{ij} V_j, i = 0, 1, \ldots, l-1. \tag{8}$$

In the literature, α and α^{-1} are swapped in the equations above. Since α and α^{-1} have the same order, we may use the form presented here. We use this form in order to make it easy to compare with the polynomial representation. For the purpose of our discussion, we will consider the case with $q = 2^n$, $m = 1$ and $l = 2^n - 1$. For a detailed discussion of the general case relation between the Lagrange Interpolation formula and the GFFT, the reader is referred to [13].

Theorem 6. *Let $F(X) = \sum_{i=0}^{2^n-1} b_i X^i$ be a function in $GF(2^n)$ with the corresponding sequence $\mathbf{v} = (v_0, v_1, \ldots, v_{2^n-2})$ where $v_i = F(\beta^i), i = 0, 1, \ldots, 2^n-2$ and $\beta \in GF(2^n)$ has order $2^n - 1$. If $F(0) = 0$ then we have*

$$b_i = \begin{cases} 0 & \text{if } i = 0 \\ V_i & \text{if } 0 < i \le 2^n - 2, \\ V_0 & \text{if } i = 2^n - 1, \end{cases} \tag{9}$$

Proof: For functions in $GF(2^n)$, the Lagrange interpolation formula can be rewritten as

$$F(X) = \sum_{i=0}^{2^n-1} b_i X^i = \sum_{\beta \in GF(2^n)} F(\beta)(1 + (X+\beta)^{2^n-1}), \tag{10}$$

where

$$b_i = \begin{cases} F(0) & \text{if } i = 0, \\ \sum_{\alpha \in GF(2^n)} F(\alpha)\alpha^{-i} & \text{if } 1 \le i \le 2^n - 1 \end{cases} \tag{11}$$

Equation (7) can be written as

$$V_i = \sum_{j=0}^{2^n-2} \beta^{-ij} v_j = \sum_{j=0}^{2^n-2} \beta^{-ij} F(\beta^j) = \sum_{\alpha \in GF^*} \alpha^{-i} F(\alpha), \tag{12}$$

where $GF^* = GF(2^n) - \{0\}$. With the convention $0^t = 1$ for any integer t, if $F(0) = 0$, then

$$\sum_{\alpha \in GF^*} \alpha^{-i} F(\alpha) = \sum_{\alpha \in GF(2^n)} \alpha^{-i} F(\alpha). \tag{13}$$

From Equation (11) and (12) we get

$$b_i = V_i, 0 < i \le 2^n - 2. \tag{14}$$

The result for $i = 2^n - 1$ follows by noting that

$$V_0 = \sum_{\alpha \in GF^*} F(\alpha), \tag{15}$$

and

$$b_{2^n-1} = \sum_{\alpha \in GF(2^n)} F(\alpha)\alpha^{-(2^n-1)} = \sum_{\alpha \in GF(2^n)} F(\alpha) = V_0 \tag{16}$$

which completes the proof. □

If $F(0) \ne 0$, then we can compute its polynomial representation by first computing the polynomial representation of the function G, where $G(X) = 0$ for $X = 0$ and $G(X) = F(X)$ otherwise. If we assume that $F(X) = \sum_{i=0}^{2^n-1} d_i X^i$ and $G(X) = \sum_{i=0}^{2^n-1} b_i X^i$ and by noting that we can express $F(X)$ as

$$F(X) = G(X) + F(0)(1 + X^{2^n-1}), \tag{17}$$

then we have

$$d_i = \begin{cases} F(0) & \text{if } i = 0, \\ b_i & \text{if } 0 < i < 2^n - 1, \\ b_{2^n-1} + F(0) & \text{if } i = 2^n - 1, \end{cases} \tag{18}$$

3.2 Relation between Boolean functions and its Galois filed polynomial representation

Let $F_2 = GF(2)$ and $F_2^{(n)} = \{x_1, \ldots, x_n | x_i \in F_2\}$. Let $f(x_1, \ldots, x_n)$ be a function from F_2^n to $F_2^{(n)}$. Then $f(x_1, \ldots, x_n)$ can be written as $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$, where y_j is a Boolean function in n variables, i.e., $y_j = y_j(x_1, \ldots, x_n)$. Since $F_2^{(n)}$ is isomorphic to $GF(2^n)$, then $f(x_1, \ldots, x_n)$ can be regarded as a function F from $GF(2^n)$ to $GF(2^n)$.

It is well known that applying a linear transformation to a function f doesn't change its nonlinear degree. It is also known that the nonlinear degree of the function $f(X) = X^d$ is $wt(d)$. The following theorem illustrates the effect of applying a linear transformation to the output coordinates of f on the coefficients of its corresponding polynomial.

Theorem 7. *Let $F(X) = X^d$ be a function of $GF(2^n)$ which corresponds to the Boolean mapping $f(x_1, \ldots, x_n) = (f_1(x), \ldots, f_n(x))$ over $F_2^{(n)}$. Then the function $G(X)$ corresponding to the Boolean mapping obtained by applying a linear transformation to the output coordinates of $f(x_1, \ldots, x_n)$ can be expressed as $G(X) = \sum_{i=0}^{2^n-1} b_i X^i$, where $b_i = 0 \forall i \notin C_d$ and C_d is the cyclotomic coset* $\pmod{2^n - 1}$.

Proof: Using Lemma 3, $G(X)$ can be expressed as

$$G(X) = \sum_{i=0}^{n-1} (a_i F(X))^{2^i} = \sum_{i=0}^{n-1} (a_i X^d)^{2^i} = \sum_{i=0}^{n-1} a_i^{2^i} X^{d2^i}. \tag{19}$$

The Theorem follows directly by noting that $X^{d2^i} = X^{(d2^i) mod (2^n-1)}$ for $X \in GF(2^n)$. □

Similarly, one can show that if $F(X) = \sum_{i \in I} a_i X^i$, then $G(X) = \sum_{j \in J} b_j X^j$ where J is the set of cyclotomic cosets modulo $2^n - 1$ corresponding to the set I.

Example 1. Consider the Boolean mapping $f(x)$ in the Table 2. Assuming $GF(2^4)$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f(x)$	0	1	4	5	9	8	13	12	15	14	11	10	6	7	2	3
$g(x)$	0	2	4	6	10	8	14	12	15	13	11	9	5	7	1	3

Table 2.

is constructed using the irreducible polynomial X^4+X^3+1, we have $F(X) = X^2$. Let $g(x)$ be the function obtained from $f(x)$ by swapping the least significant bits of the output. I.e., $g(x_1, x_2, x_3, x_4) = (f_1(x), f_2(x), f_4(x), f_3(x))$, then we have $G(X) = 2X + 10X^2 + 6X^4 + 12X^8$.

The following theorem illustrates the effect of applying a linear transformation to the input coordinates of a given Boolean function on the coefficients of its corresponding polynomial.

Theorem 8. *Let $F(X) = X^d$ be a function of $GF(2^n)$ which corresponds to the Boolean mapping $f(x_1, \dots, x_n) = (f_1(x), \dots, f_n(x))$ over $F_2^{(n)}$. Let $G(x)$ be the function which corresponds to the Boolean mapping obtained by applying a linear transformation to the input coordinates of $x_1, \dots, x_n$ while fixing $f(x_1, \dots, x_n)$. Then $G(X)$ can be expressed as $G(X) = \sum_{i=0}^{2^n-1} b_i X^i$, $b_i = 0$ for $wt(i) > wt(d)$, where $wt(d)$ denotes the Hamming weight of d.*

Proof: Using Lemma 3, $G(X)$ can be expressed as

$$G(X) = (\sum_{i=0}^{n-1} c_i X^{2^i})^d \tag{20}$$

Let $d = \sum_{j=0}^{n-1} d_j 2^j$ and let J denote the set $\{j_1, \dots, j_s\}, s = wt(d)$, for which $d_j = 1$. Then we have

$$G(X) = \prod_{j \in J} (\sum_{i=0}^{n-1} c_i X^{2^{i+j}}) \tag{21}$$

$$= (\sum_{i_1=0}^{n-1} c_{i_1} X^{2^{i_1+j_1}})(\sum_{i_2=0}^{n-1} c_{i_2} X^{2^{i_2+j_2}}) \dots (\sum_{i_1=0}^{n-1} c_{i_s} X^{2^{i_1+j_s}}) \tag{22}$$

$$= \sum_{i_1, i_2, \dots, i_s} c_{i_1} c_{i_2} \dots c_{i_s} X^{2^{i_1+j_1} + 2^{i_2+j_2} + \dots + 2^{i_s+j_s}} \tag{23}$$

The Theorem follows by noting that $wt(2^{i_1+j_1} + \dots + 2^{i_s+j_s}) = s \leq wt(d)$. $\square$

Let $W = \max_{i \in I} wt(i)$. Then one can show that if $F(X) = \sum_{i \in I} a_i X^i$, then $G(X) = \sum_{j \in J} b_j X^j$ where J is the set of elements with Hamming weight $\leq W$.

The following theorem illustrates the effect of changing the irreducible polynomial used to construct the finite field on the coefficients resulting polynomial.

Theorem 9. *Let $F(X)$ be a function of $GF(2^n)$ which corresponds to the Boolean mapping $f(x_1, \dots, x_n) = (f_1(x), \dots, f_n(x))$ over $F_2^{(n)}$ using irreducible R_1. Then the function $G(x)$ which corresponds to the boolean mapping $f(x_1, \dots, x_n)$ and constructed using a different irreducible polynomial $R_2 \neq R_1$ can be expressed as*

$$G(X) = L(F(L^{-1}(X))), \tag{24}$$

where L is an invertible linear transformation over $GF(2^n)$.

Proof: Consider the finite field generated by an irreducible polynomial $R_1(X)$. In this case, $GF(2^n) = F_2[X]/(R_1(X)) = \{\sum_{i=0}^{n-1} c_i X^i | c_i \in F_2\}$ where the multiplication is performed by modulus $R_1(X)$. Then every element in the field can be expressed as $\sum_{i=0}^{n-1} a_i \alpha^i$ where $a_i \in GF(2)$ and α is a root of $R_1(X)$. Similarly, if the field was generated using an irreducible polynomial $R_2(X)$. In this case, $GF(2^n) = F_2[X]/(R_2(X)) = \{\sum_{i=0}^{n-1} c_i X^i | c_i \in F_2\}$ where the multiplication is performed by modulus $R_2(X)$. In this case, every element in the field can be expressed as $\sum_{i=0}^{n-1} b_i \beta^i, b_i \in GF(2)$ where β is a root of $R_2(x)$. However, we can express β^i as

$$\beta^i = \sum_{j=0}^{n-1} a_j \alpha^j, a_j \in GF(2), 0 \leq i < n. \tag{25}$$

This means that we can write $G(X) = L(F(L^{-1}(X))$ where $L(.)$ is the linear transformation used to convert between the α and the β basis. □

From the theorem above changing the irreducible polynomial is equivalent to applying a linear transformation to both the input and the output coordinates, and hence we have the following corollary

Corollary 10. *Let $F(X) = \sum_{i \in I} a_i X^i$ be a function of $GF(2^n)$ which corresponds to the Boolean mapping $f(x_1, \ldots, x_n) = (f_1(x), \ldots, f_n(x))$ over $F_2^{(n)}$ using irreducible R_1. Let the $W = \max_{i \in I} wt(i)$. Then the function $G(x)$ corresponds to the boolean mapping $f(x_1, \ldots, x_n)$ and constructed using a different irreducible polynomial $R_2 \neq R_1$ can be expressed as*

$$G(X) = \sum_{j \in J} b_j X^j, \tag{26}$$

where J is the set of elements with Hamming weight $\leq W$.

Example 2. Consider the Boolean function described in Table 3.

x	0	1	2	3	4	5	6	7
$f(x)$	0	1	3	4	5	6	7	2

Table 3.

Using the irreducible polynomial $X^3 + X^2 + 1$ with root β, we have $F(X) = 2X + 2X^2 + 3X^3 + 4X^4 + X^5 + 7X^6$. Now, consider the irreducible polynomial $X^3 + X + 1$ with root α. One can prove that $\beta = \alpha^3$. Thus we have the following linear transformation

$$\begin{pmatrix} 1 \\ \beta \\ \beta^2 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{pmatrix} 1 \\ \alpha \\ \alpha^2 \end{pmatrix} \tag{27}$$

Applying this linear transformation to both the input and the output of the truth table we get $L^{-1}(x) and L(f(x))$ in Table 4. Interpolating the relation between $L^{-1}(x)$ and $L(f(x))$, we get $L(F(X)) = (L^{-1}(X))^3$.

x	0	1	2	3	4	5	6	7
$L^{-1}(x)$	0	1	3	2	5	4	6	7
$f(x)$	0	1	3	4	5	6	7	2
$L(f(x))$	0	1	2	5	4	6	7	3

Table 4.

To summarize the results in this section, a linear transformation on the output coordinates affects only the coefficients of the exponents that belong to the same cyclotomic cosets of the exponent in the original function representation. A linear transformation on the input coordinates or changing the irreducible polynomial affect only the coefficients of the exponents with Hamming weight less than or equal to the maximum Hamming weight of the exponents in original function representation.

4 Checking algebraic expressions for trap doors

In [5] the authors presented a method to construct trap door block ciphers which contains some hidden structures known only to the cipher designers. The sample trapdoor cipher in [5] was broken [12] and designing practical trape door S-boxes is still an intersting topic. In this section we discuss how to check if the S-boxes or the round function has a simple algebraic structure. In particular, we consider the case where we can represent the round function or the S-boxes by a monomial. The number of invertible linear transformations grows exponentially with n. Using exhaustive search to check if applying an invertible linear transformation to the output and/or the input coordinates of the Boolean function $f(x_1, \ldots, x_n) = (f_1(x), \ldots, f_n(x))$ leads to a simpler polynomial representation becomes computationally infeasible even for small values of n. In this section we show how to check for the existence of such simple description. Note that we only consider the case of polynomials over $GF(2^n)$. S-boxes with a complex algebraic expression over $GF(2^n)$ may have a simpler description over other fields.

4.1 Undoing the effect of a linear transformation on the output coordinates

First, we will consider the case of a function $G(X)$ obtained by applying a linear transformation of the output coordinates of a monomial function X^d. The algebraic description of such a function will have nonzero coefficients only for exponents $\in C_d \pmod{2^n - 1}$. Thus $G(X)$ is expressed as

$$G(X) = \sum_{i=0}^{2^n-1} b_i X^{2^i d}, \tag{28}$$

$b_i = 0$ if $i \notin C_d$. A linear transformation of the output coordinates of $G(X)$ can be expressed as

$$L(G(X)) = \sum_{j=0}^{n-1} a_j \Big(\sum_{i=0}^{2^n-1} b_i X^{2^i d} \Big)^{2^j} \tag{29}$$

$$= \sum_{j=0}^{n-1} a_j \sum_{i=0}^{2^n-1} {b_i}^{2^j} X^{(2^{i+j})d} \tag{30}$$

By equating the coefficients of X^i to zero except for $i = d$, the above equation forms a system of $n \times n$ linear equations (with unknowns $a_i^{\prime}s \in GF(2^n)$) which can be checked for the existence of a solution using simple linear algebra.

Example 3. Let $G(X) = 2X + 10X^2 + 6X^4 + 12X^8, X \in GF(2^4)$ constructed using the irreducible polynomial $X^4 + X^3 + 1$, Suppose we want to check if there exists a linear transformation on the output coordinates of $G(X)$, $L(G(X))$ such that the resulting polynomial has only one term with degree 2. Using the theorem above, form the set of 4×4 linear equations over $GF(2^4)$ we get:

$$\begin{bmatrix} b_0 & b_3^2 & b_2^4 & b_1^8 \\ b_1 & b_0^2 & b_3^4 & b_2^8 \\ b_2 & b_1^2 & b_0^4 & b_3^8 \\ b_3 & b_2^2 & b_1^4 & b_0^8 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \tag{31}$$

For $G(X)$ above we have $b_0 = 2, b_1 = 10, b_2 = 6, b_3 = 12$. Thus

$$\begin{bmatrix} 2 & 6 & 7 & 11 \\ 10 & 4 & 13 & 12 \\ 6 & 11 & 9 & 7 \\ 12 & 13 & 10 & 14 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{32}$$

Solving for a_i's we get

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 10 \\ 6 \\ 12 \\ 2 \end{pmatrix} \tag{33}$$

and $L(G(X)) = 10G(X) + 6G(X)^2 + 12G(X)^4 + 2G(X)^8 = X^2$

4.2 Undoing the effect of a linear transformation on the input coordinates

Consider a function $G(X)$ obtained by applying a linear transformation to the input coordinates of a monomial function X^d. The algebraic description of such a function will have zero coefficients for all exponents with Hamming weight $> d$. Thus $G(X)$ is expressed as

$$G(X) = \sum_{i=0}^{2^n-1} b_i X^i, \tag{34}$$

$b_i = 0$ if $wt(i) > d$

A linear transformation of the input coordinates of $G(X)$ can be expressed as

$$L(G(X)) = \sum_{i=0}^{2^n-1} b_i \sum_{j=0}^{n-1} (a_j X^{2^j})^i \tag{35}$$

If one tries to evaluate the above expression and equate the coefficients to the coefficients of a monomial, then one has to solve a set of non linear equations with unknowns $a_j, j = 0, 1, \ldots, n-1$.

To overcome this problem, we will reduce the problem of undoing the effect of a linear transformation on the input coordinates to undoing the effect of a linear transformation on the output coordinates.

Consider $G(X)$ obtained by a linear transformation on the input coordinates of $F(X)$. Then $G(X) = F(L(X))$. Thus we have $G^{-1}(X) = L^{-1}(F^{-1}(X))$. If $F(X)$ is a monomial, then $F^{-1}(X)$ is also a monomial and our problem is reduced to finding the linear transformation L^{-1} on the output coordinates of $F^{-1}(X)$ which is equivalent to solving a system of linear equations in n variables.

Example 4. Consider the function $G(X) = 8X^2 + 9X^3 + X^4 + 11X^5 + 14X^6 + X^7 + 12X^8 + 2X^9 + 9X^{10} + 4X^{11} + 11X^{12} + 14X^{13} + 14X^{14} \in GF(2^4)$ where $GF(2^4)$ is constructed using the irreducible polynomial $X^4 + X^3 + 1$. In this case, we have $G(X)^{-1} = 5X^7 + 5X^{11} + 11X^{13} + 15X^{14}$. In this case, we have 60 linear transformations on the output coordinates of $G^{-1}(X)$ that will map it to a monomial of exponent with weight 3. Out of these 60 transformations, we have 15 linear transformations such that $L(G^{-1}(X)) = aX^{13}, a \in GF(2^4)$. In particular, the linear mapping $L(X) = X + 14X^2 + 9X^4 + 14X^8$ on the output bits of $G^{-1}(X)$ reduces $G^{-1}(X)$ to X^{13}, i.e., $L(G^{-1}(X)) = X^{13}$ and hence $G(X) = (L(X))^7$.

Undoing the effect of changing the irreducible polynomial corresponds to undoing the effect of a linear transformation on both the input and the output coordinates which seems to be a hard problem. The number of irreducible polynomials of degree n over a finite field with q elements is given by

$$I_n = \frac{1}{n} \sum_{d|n} \mu(d) q^{n/d}, \tag{36}$$

where $\mu(d)$ is defined by

$$\mu(d) = \begin{cases} 1 & \text{if } d = 1, \\ (-1)^k & \text{if } d \text{ is the product of } k \text{ distinct primes,} \\ 0 & \text{if } d \text{ is divisible by the square of a prime.} \end{cases} \tag{37}$$

Since the dominant term in I_n occurs for $d = 1$, we get the estimate

$$I_n \approx \frac{q^n}{n} \tag{38}$$

Thus for typical S-box sizes, exhaustive search through all the set of $(2^n/n)$ irreducible polynomials seems to be a feasible task.

References

1. R. Lidl and H. Niederreiter, *Finite Fields (Encyclopedia of Mathematics and its Applications)* , Addison Wesley. Reading, MA. 1983.
2. R. J. McEliece, *Finite Fields For Computer Scientists and Engineers* , Kluwer Academic Publishers. Dordrecht. 1987.
3. T. Jakobsen and L. Knudsen, *The Interpolation Attack on Block Ciphers, LNCS 1267*, Fast Software Encryption. pp. 28-40. 1997.
4. T. Jakobsen, *Cryptanalysis of Block Ciphers with Probabilistic Non-linear Relations of Low Degree*, Proceedings of Crypto'99. LNCS 1462. pp. 213-222. 1999.
5. V. Rijmen and B. Preneel, *A family of trapdoor ciphers*, Proceedings of Fast Software Encryption. LNCS 1267. pp. 139-148. 1997.
6. M. Sudan, *Decoding Reed Solomon Codes beyond the error-correction bound*, Journal of Complexity. Vol. 13. no 1. pp180-193. March, 1997.
7. G. Gong and S. W. Golomb, *Transform Domain Analysis of DES*, IEEE transactions on Information Theory. Vol. 45. no. 6. pp. 2065-2073. September, 1999.
8. K. Nyberg and L. Knudsen, *Provable Security Against a Differential Attack*, Journal of Cryptology. Vol. 8. no. 1. 1995.
9. K. Aoki, *Efficient Evaluation of Security against Generalized Interpolation Attack*, Sixth Annual Workshop on Selected Areas in cryptography SAC'99. Workshop record. pp. 154-165. 1999.
10. S.W. Golomb,*Shift Register Sequences*, Aegean Park Press. Laguna Hills, California. 1982.
11. R.E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley. Reading, MA. 1990.
12. H. Wu, F. Bao, R. Deng and Q. Ye *Cryptanalysis of Rijmen-Preneel Trapdoor Ciphers, LNCS 1514*, Asiacrypt'98. pp. 126-132. 1998.
13. G. Gong and A.M. Youssef, *Lagrange Interpolation Formula and Discrete Fourier Transform* , Technical Report. Center for Applied Cryptographic Research. University of Waterloo. 1999.

Stochastic Cryptanalysis of Crypton

Marine Minier and Henri Gilbert

France Télécom R&D
38-40, rue du Général Leclerc
92794 Issy les Moulineaux Cedex 9 - France
Tel: +33 1 45 29 44 44

Abstract. Crypton is a 12-round blockcipher proposed as an AES candidate by C.H. Lim in 1998. In this paper, we show how to exploit some statistical deficiencies of the Crypton round function to mount stochastic attacks on round-reduced versions of Crypton. Though more efficient than the best differential and linear attacks, our attacks do not endanger the practical security offered by Crypton.

1 Introduction

Crypton [Li98] is a 12-round blockcipher which was submitted by C.H. Lim as one of the 15 candidates at the first Advanced Encryption Standard conference in August 1998. Crypton offers several interesting features. The encryption and decryption processes are strictly identical up to the key schedule (a quite remarkable property given the substitution/permutation structure of the cipher). Crypton is highly parallelizable and flexible, and thus well suited for efficient implementation on nearly any hardware or software platform. Moreover, Crypton provides some provable resistance against linear and differential cryptanalysis.

The main cryptanalytic results obtained on Crypton so far are the analysis of the best differential and linear attacks by the algorithm designer [Li98], a transposition of the square attack to the 6-round Crypton by C. D'Halluin et al. [Hal99], the discovery of some weak keys by Vaudenay [Ba99], and statistical observations contained in an annex of [Ba99].

C.H. Lim introduced in 1999 [Li99] a modified Crypton (denoted by Crypton v1.0) with a new keyschedule and new S-boxes designed as to lower the number of high probability differential and linear characteristics. Though most of this paper is dedicated to the analysis of the initial version of Crypton, the impact of the Crypton v1.0 S-box modifications is also discussed in the last Section.

We present here two attacks of round reduced versions of Crypton (up to 8 rounds) which are based on iterative statistical properties of the round function. A short outline of preliminary (unquantified) versions of these attacks has been already published in a paper presented at the second AES conference [Ba99]. Based on extra analysis and computer experiments, we provide here more precise assessments of the statistical biases and the performance of these attacks.

Both attacks can be broadly described as stochastic attacks. The block values (or a subset of the difference values if the attack uses differential statistics) are partitioned into a small number of classes, and any k-round partial encryption is modeled as a stochastic process (i.e. a sequence of random variables providing the class values at the boundaries of the various rounds). Each round is characterized by one matrix of key dependent or key-independent transition probabilities between input and output classes. Under the heuristic assumption that the above process is nearly markovian, the behaviour of such a k-rounds partial encryption is well approximated by the product of the k one-round transition probabilities matrices [1].

We do not claim that the idea of that kind of generalization of more traditional "characteristics-based" attacks is new : Murphy et al.' likehood estimation [Mu], Vaudenay's χ^2 cryptanalysis[Va95], Lai and Massey's modeling of markovian ciphers [LM91], Harpes and Massey's partition cryptanalysis [HM97]provide frameworks which describe similar generalizations. However, there are not yet numerous examples of ciphers where such approaches bring some real added value. We believe Crypton is a good example of an algorithm for which this is the case.

A stochastic attack is feasible if there exists a partition of the blocks (or of the considered subset of difference values) such that for each key value, the transition probabilities among classes differ substantially from the transition probabilities a random permutation of the block or difference values would provide. The key cryptanalytic issue consists in finding such a suitable partition. In the case of Crypton, the partitions of the block values and of difference values we are using are based upon some invariance properties of the linear part of the round function which involve only four "active" bytes of the inputs or outputs to the non linear part.

The rest of this paper is organized as follows : Section 2 briefly summarizes the Crypton cipher. Section 3 presents the iterative statistical properties which form the starting point for our attacks. Section 4 presents a stochastic attack based upon a partition of difference values into 257 classes. Section 5 presents a stochastic attack based upon a partition of blocks into 16 classes. Section

[1] Stochastic attacks represent a generalization of "Characteristics-based attacks", such as linear attacks, differential attacks, or truncated differential attacks. As a matter of fact, characteristics based attacks are based upon a partition of block or difference values at each round into only two classes. For instance, in linear attacks, blocks are partitioned according to the binary value of a fixed linear combination of the key bits. In differential attacks one considers the unbalanced partition of the differences between one single difference value on one hand and the complementary set of all other difference values on the other hand. In truncated differential attacks, one considers a partition of difference values according to the characteristic function of a set of difference values satisfying certain constraints, etc. In characteristics based attacks of blockciphers, one single transition probability, namely the probability of the considered characteristic, entirely determines the (2x2) transition probabilities matrix associated with a partial encryption. The stochastic attacks considered in this paper are not "characteristics-based" because they involve partitions in strictly more than two classes.

6 investigates the (quite positive) impact of the modifications introduced in Crypton v1.0 and Section 7 concludes the paper.

2 An Outline of Crypton

Crypton encrypts 128-bit blocks under the control a key of length up to 256 bits. The nominal value of the r number of rounds is 12. The algorithm consists of the encryption function itself and a keyschedule that derives $(r+1)$ 128-bit subkeys from the key and an encryption function. Since the attacks presented here do not rely at all upon the properties of the key schedule, we do not describe it here.

The encryption function consists of r rounds surrounded by an input transformation (defined by an XOR between the plaintext block and the first subkey) and an output transformation (defined as a fixed modulo 2 linear mapping).

Let us represent a 128-bit block A by :

$$A = \begin{pmatrix} a_{0,3} & a_{0,2} & a_{0,1} & a_{0,0} \\ a_{1,3} & a_{1,2} & a_{1,1} & a_{1,0} \\ a_{2,3} & a_{2,2} & a_{2,1} & a_{2,0} \\ a_{3,3} & a_{3,2} & a_{3,1} & a_{3,0} \end{pmatrix} \begin{array}{c} A[0] \\ A[1] \\ A[2] \\ A[3] \end{array}$$

where each $a_{i,j}$ is a byte.
One round consists of a byte substitution γ, followed by a bit permutation π, a bytes transposition τ and a subkey addition σ. γ (γ_o for odd round, γ_e for even round) uses two S-boxes S_0 and S_1. We only describe of γ_o ; to obtain γ_e one just needs to exchange S_0 and S_1.

$$\begin{pmatrix} b_{0,3} & b_{0,2} & b_{0,1} & b_{0,0} \\ b_{1,3} & b_{1,2} & b_{1,1} & b_{1,0} \\ b_{2,3} & b_{2,2} & b_{2,1} & b_{2,0} \\ b_{3,3} & b_{3,2} & b_{3,1} & b_{3,0} \end{pmatrix} \overset{\gamma_o}{\leftarrow} \begin{pmatrix} S_1(a_{0,3}) & S_0(a_{0,2}) & S_1(a_{0,1}) & S_0(a_{0,0}) \\ S_0(a_{1,3}) & S_1(a_{1,2}) & S_0(a_{1,1}) & S_1(a_{1,0}) \\ S_1(a_{2,3}) & S_0(a_{2,2}) & S_1(a_{2,1}) & S_0(a_{2,0}) \\ S_0(a_{3,3}) & S_1(a_{3,2}) & S_0(a_{3,1}) & S_1(a_{3,0}) \end{pmatrix}$$

π (π_o for odd rounds, π_e for even rounds) is a bit permutation. We only describe effects of π_o for odd rounds, effects of π_e are similar. π_o is given by

$$\begin{aligned} T &= A[0] \oplus A[1] \oplus A[2] \oplus A[3], \\ B[0] &\leftarrow (A[0] \wedge MI_0) \oplus (A[1] \wedge MI_1) \oplus (A[2] \wedge MI_2) \oplus (A[3] \wedge MI_3) \oplus T, \\ B[1] &\leftarrow (A[0] \wedge MI_1) \oplus (A[1] \wedge MI_2) \oplus (A[2] \wedge MI_3) \oplus (A[3] \wedge MI_0) \oplus T, \\ B[2] &\leftarrow (A[0] \wedge MI_2) \oplus (A[1] \wedge MI_3) \oplus (A[2] \wedge MI_0) \oplus (A[3] \wedge MI_1) \oplus T, \\ B[3] &\leftarrow (A[0] \wedge MI_3) \oplus (A[1] \wedge MI_0) \oplus (A[2] \wedge MI_1) \oplus (A[3] \wedge MI_2) \oplus T, \end{aligned}$$

where $MI_0 = c0300c03, MI_1 = 03c0300c, MI_2 = 0c03c030, MI_3 = 300c03c0$ (in hexadecimal). We can notice that if we omit the addition with T, π results in permutations of 4 2-bit words in each of the 16 2-bit words columns of the A matrix.

The bytes transposition τ just consists in exchanging all $a_{i,j}$ and $a_{j,i}$ pairs of bytes in the A matrix, and the key addition σ_K just consists of an exclusive-or between A and a 128-bit subkey K.

3 Statistical Properties of the Round Function

In this Section we introduce two iterative properties of the $\tau \circ \pi$ linear part of the Crypton round function which involve only four bytes of the inputs or outputs to the non linear part - and thus represent limitations in the diffusion achieved by $\tau \circ \pi$. These two properties, which are outlined in [Ba99] are to some extent dual of each other, and can be summarized as follows : (1) some $\tau \circ \pi$ input values equal to zero except on at most four bytes are transformed into output values equal to zero except for at most four other bytes ; (2) for certain sets of 4 of the 16 $\tau \circ \pi$ input bytes and certain associated sets of 4 of the 16 $\tau \circ \pi$ output bytes, there exist a (linear) four bytes to 4 bits function Φ such that the images by Φ of the four input bytes and the four output bytes are equal. Property (1), as applied to difference values in the encryption of pairs of plaintexts, can be seen as an iterative truncated differential whereas property (2) can be to a certain extent compared to a set of iterative linear characteristics.

We introduce some additional notation to split each $a_{i,j}$ byte of an A block into four 2-bit words : $a_{i,j} = (a_{3,i,j}, a_{2,i,j}, a_{1,i,j}, a_{0,i,j})$ where i is the line index and j the column index. $i \in [0,3]$ and $j \in [0,3]$. $a_{k,i,j}$ will be the 2-bits word of line i, column j and position k. We define the "square" associated with the (k,i,j) triplet as the $((a_{k,i,j}, a_{k,i+2,j}, a_{k,i,j+2}, a_{k,i+2,j+2}))$ quartet of 2-bit words. Note that indexes are implicitly taken modulo 4. Under some conditions, squares are preserved (up to a modification of the associated (k,i,j) indexes) by the γ_o, γ_e, π_o, π_e and τ functions.

3.1 Property (1)

One can see that if we omit T, π_o and π_e just permute the $a_{k,i,j}$ 2-bit words (more precisely, they only permute the j indexes), and thus they transform any square associated to a (k,i,j) triplet into the same square associated with another (k,i',j') triplet. This stays valid for the real π_o and π_e functions under the two additional conditions (i) $a_{k,i,j} = a_{k,i+2,j}$ and (ii) $a_{k,i,j+2} = a_{k,i+2,j+2}$. Squares are also preserved by the τ function (up to a modification of the i and j indexes). Moreover, under conditions (i) and (ii), "twin squares" associated with two (k,i,j) and $(k+2,i,j)$ triplets of indexes are transformed into two other "twin squares" associated with two (k,i',j') and $(k+2,i',j')$ triplets of indexes.

In order to cryptanalytically exploit property (1), we can consider special difference values equal to zero except on two "twin squares" (k,i,j) and $(k+2,i,j)$ and such that the (i) and (ii) conditions are satisfied for both squares. For instance, squares of difference values of the following form stay entirely invariant under the π_o mapping :

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 00x_1x_200y_1y_2 & 0 & 00x'_1x'_200y'_1y'_2 \\ 0 & 0 & 0 & 0 \\ 0 & 00x_1x_200y_1y_2 & 0 & 00x'_1x'_200y'_1y'_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & a & 0 & b \\ 0 & 0 & 0 & 0 \\ 0 & a & 0 & b \end{pmatrix}$$

Moreover, for the above example, τ just modifies location of squares :

$$\begin{array}{|c|c|c|c|}\hline & & & \\ \hline & a & & b \\ \hline & & & \\ \hline & a & & b \\ \hline \end{array} \overset{\tau}{\Rightarrow} \begin{array}{|c|c|c|c|}\hline b & & b & \\ \hline & & & \\ \hline a & & a & \\ \hline & & & \\ \hline \end{array}$$

In the sequel we will keep using a representation of twin squares at the byte level (as in the above representation of τ) to more compactly depict the effects of π_o, π_e and τ.

3.2 Property(2)

Let us consider any two twin squares with a (k, i, j) triplet and a $(k + 2, i, j)$ triplet of indexes. As seen before, if there was no T term in the π definition, these two twin squares would be just displaced by the $\tau \circ \pi$ linear function (without any change in the quartet values) to two twin squares associated with (k, i', j') and $(k + 2, i', j')$ triplets of indexes. Now if we take the T term into account, we can see that the $\phi_{k,i',j'}$ 2-bit XOR of the four 2-bit words of the (k, i', j') output square is still equal to the $\phi_{k,i,j}$ XOR of the four 2-bit words of the (k, i, j) input square (just because the additional terms introduced by T twowise compensate). The same property obviously holds for the squares associated with the $(k+2, i, j)$ input triplet and the $(k + 2, i', j')$ output triplet : $\phi_{k+2,i,j} = \phi_{k+2,i',j'}$. Thus, if we denote by $\Phi_{k,i,j}$ the 4-bit word $\phi_{k,i,j} \oplus 4.\phi_{k+2,i,j}$, we can summarize the obtained invariance property by the equality $\Phi_{k,i,j} = \Phi_{k,i',j'}$. In other words, the 4-bit linear combination $\Phi_{k,i,j}$ of the four bytes involved in two twin squares is kept invariant by the linear part of Crypton (up to a change of considered four bytes positions).

In Section 5, we will mount an attack based upon partitioning block values in 16 classes according to the value of such a 4-bit Φ values.

In summary, we have identified in properties (1) and (2) some correlations between the input and the output of the $\tau \circ \pi$ part of the round function which involve only 4 ”active” input and output bytes. It remains to study how much correlation is left on entire rounds if one also takes the non linear part of the Crypton round function into account.

4 Stochastic Attack Using Differential Properties

4.1 Computing Transition Probabilities

The cryptanalysis is based upon property (1) and uses differences of the form described in part 3.1. That’s why, in order to describe elements which stay invariant by π and τ, we introduce two sets of possible difference values at the byte level :

$$D_1 = \{0,1,2,3,16,17,18,19,32,33,34,35,48,49,50,51\}$$
$$D_2 = \{0,4,8,12,64,68,72,76,128,132,136,140,192,196,200,204\}$$

We include zero in both sets although in practice this difference value can't appear in an attack. D_1 corresponds to all possible choices of the 00xx00xx byte and D_2 to all possible choices of the xx00xx00 byte. Those sets permit us to create "twin square" invariant under the linear part. We denote by (δ_1, δ_2) differences of the form

$$\Delta_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \delta_1 & 0 & \delta_2 \\ 0 & 0 & 0 & 0 \\ 0 & \delta_1 & 0 & \delta_2 \end{pmatrix}$$

where Δ_1 is represented by a 4×4 matrix of bytes with $\delta_1 \in D_1$ and $\delta_2 \in D_1$ or of the form

$$\Delta_2 = \begin{pmatrix} 0 & \delta_1 & 0 & \delta_2 \\ 0 & 0 & 0 & 0 \\ 0 & \delta_1 & 0 & \delta_2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

with $\delta_1 \in D_2$ and $\delta_2 \in D_2$. The above Δ_1 and Δ_2 difference matrices are just examples of all possible difference values that can be taken. As a matter of fact, (δ_1, δ_2)can be any "twin square" difference satisfying conditions (i) and (ii) of Section 2.
We can say by the invariance properties seen above that there exist δ'_1 and δ'_2 in D_1 such that

$$\tau(\pi_i(\Delta_1)) = \begin{pmatrix} \delta'_2 & 0 & \delta'_2 & 0 \\ 0 & 0 & 0 & 0 \\ \delta'_1 & 0 & \delta'_1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

So we can deduce that with a certain probability p, the "twin squares" represented by the (δ'_1, δ'_2) pair could result, after being passed through S-boxes, into a (δ_3, δ_4) pair such that

$$\gamma_i(\tau(\pi_i(\Delta_1))) = \begin{pmatrix} \delta_3 & 0 & \delta_4 & 0 \\ 0 & 0 & 0 & 0 \\ \delta_3 & 0 & \delta_4 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

An S-box output with this form permits us to use another time the invariance property of $\gamma_i(\tau(\pi_i(\Delta_1)))$ into π_i and τ on an other "twin square". We obtain, with a certain probability p', the invariance of Δ_1 on one entire round. With the same construction, we can establish the invariance of Δ_2 on one round with a certain probability p".
For all (δ_1, δ_2) pairs of D_1 and all (δ_3, δ_4) pairs of D_1, we can compute the probability of getting a (δ_3, δ_4) output difference from a (δ_1, δ_2) input difference after

one round. All probabilities are key independent and only depend on differential properties of S-boxes.

$$\begin{aligned} Pr[\Delta' = (\delta_3, \delta_4) | \Delta = (\delta_1, \delta_2)] &= Pr[\delta_1 \to \delta_3].Pr[\delta_2 \to \delta_3] \\ &\quad .Pr[\delta_1 \to \delta_4].Pr[\delta_2 \to \delta_4] \\ &= \frac{d_{\delta_1\delta_3}}{256}.\frac{d_{\delta_2\delta_3}}{256}.\frac{d_{\delta_1\delta_4}}{256}.\frac{d_{\delta_2\delta_4}}{256} \end{aligned}$$

where $d_{\delta_i\delta_j}$ is the number of bytes such that : $\delta_j = S(x) \oplus S(x \oplus \delta_i)$, S represents the appropriate S-box of Crypton. We obtain a 256×256 matrix M of transition probabilities over one round by computing p for 256 couples of possible input values (δ_1, δ_2) and for 256 couples of possible output values (δ_3, δ_4) with $\delta_1, \delta_2, \delta_3, \delta_4 \in D_1$. Columns of M represent all probabilities of transition between input (δ_1, δ_2) and output (δ_3, δ_4).We use the same method to compute probabilities of transition associated with D_2.

Now let us consider differentials over two rounds, i.e. transition between an input value (δ_1, δ_2) and an output value (δ_3, δ_4). A lower bound on the probabilities of such differentials is provided by summing up the probabilities of all intermediate values which belong to D_1 :

$$p_{i,j} = \sum_{i=0}^{255}\sum_{j=0}^{255} p_{i,k}p_{k,j}$$

where $p_{i,j}$ represent coefficients of M. With this relation, we consider all the possible intermediate values. There exists a stochastic dependence between the difference values at the various rounds. We make the heuristic assumption that the sequence of difference values over several rounds satisfies the "Markov property", i.e. the distribution of probabilities of the differences at the output of any round only depend on the distribution of probabilities of the differences at the input of the same round. So, we can compute M^n, which represents key-independent transition probabilities between input and output differences on an n-rounds scheme. For example, to compute transition probabilities for 6 rounds, we compute M^6. One cryptanalytically meaningful measure of the unbalanceness of the obtain matrix M consists of computing the sums of the probabilities in each column. The performance of the attack hereafter depends upon the value obtained for the "best column", i.e. the best pair of input difference values (δ_1, δ_2).

4.2 Attack Procedure

We present here an attack on a complete eight-round Crypton (i.e. taking into account the first addition of subkey K_0 and the final transformation). Under the heuristic assumptions summarized above, we can compute probabilities of best couples of difference on several rounds. In particular we obtain the following figures for 6 inner rounds of Crypton : if the input difference is the (18,

18) pair of $\varDelta_1$ values, the probability that the output be a (δ_1, δ_2) pair with (δ_1,δ_2)$\in D_1 \times D_1$ is $2^{-120.72}$. In the same way, if input is the (128, 128) pair of D_2 values, probability that the output be a (δ_1, δ_2) pair of D_2 values is $2^{-112.62}$.

This result on six inner rounds can be used to attack an eight inner rounds attack. Due to the late occurrence of the σ key addition in the first inner round, we can control differences at the output of the first occurrence of γ, and thus the first round can be treated just as a keyless "initial permutation". A 1R-attack permits to also gain the eighth round. So, we can exploit the a priori known 6-rounds properties in a chosen plaintext attack to obtain some bits of information on the last round key K_8.

In order to efficiently generate pairs of chosen plaintexts which difference is equal to the $\varDelta_2 = (128, 128)$ value at the output of the first occurrence of γ, we group plaintexts in structures. Two 128-bits elements belong to the same structure if and only their images by γ are equal except for some of the 16 bits of $\varDelta_2$. Each structure has 2^{16} elements. We know that, if two X and X' plaintexts belong to the same structure, then with probability $2^{-112.62}$, the correponding inputs to the the eighth round are of the form Y and $Y \oplus \varDelta_2$ (where $\varDelta_2$ is of D_2 values). We only consider those (X, X') pairs such that the C$\oplus$C' ciphertext difference is null everywhere except at most on the four non zero bytes of the $\varDelta_2$ Y differences. For those pairs which pass that filtering condition, we go up the last round starting from C and C' and checking whether the resulting $Y \oplus Y'$ has the right form. Some couples are "false" alarms, i.e. pass the filtering condition but don't belong to our set (this happens with a probability equal to 2^{-96}). Once selected "good" couples, we test the possible values of four bytes of key in going up at the end of eighth round. The candidate value which appears most often is the right one. We obtain four bytes of information on K_8 [2].

We can go up through the final transformation $\varPhi_e$ because it does not change output values of eighth round. Taking into account the first key-addition σ_0 just increases the number of "false" alarms. The number of plaintexts to cipher is $N = 2^{112.62}$. But we must take a security for "false" alarms. We claim that taking $N = 2^{114.62}$ is enough. So we can obtain 32 bits of information of K_8 by ciphering $2^{114.62}$ couples X and X$\oplus\varDelta$ in a complete eight-rounds version of Crypton. Complexity of this attack is $2^{114.62}$ encryptions and 2^{96} additional computations.

This attack is faster than an exhaustive search and than all differential attacks. As a matter of fact, it can be shown that the probability of the best characteristic for an eigth-round attack is 2^{-120}.

[2] The same procedure can be repeated with other square locations (at the expanse of a slight increase of the N number of chosen plantexts) to entirely derive K_8. Once the last subkey has been entirely derived, the same procedure can be repeated (with the same plaintexts) to derive the entire expanded key.

5 Stochastic Cryptanalysis of Crypton Using a Partition of Blocks in 16 Classes

5.1 Computation of Transition Probabilities

The cryptanalysis presented in this Section is based on Property (2) of Section 2. Let us consider inside an intermediate block

$$A = \begin{pmatrix} * & * & * & * \\ * & X & * & Y \\ * & * & * & * \\ * & Z & * & T \end{pmatrix}$$

encountered in the Crypton encryption process, a (X, Y, Z, W) quartet of bytes associated with a given (i, j) pair of indices. We can partition the blocks space into 16 classes according to the 4-bit value $\Phi_0[X, Y, Z, W] = \Phi_{0,i,j}$ (or alternatively into 16 other classes according to the 4-bit value $\Phi_1[X, Y, Z, W] = \Phi_{1,i,j}$).

Property (2) states that the linear part of Crypton leaves Φ_0 and Φ_1 values unchanged (provided that the final Φ_0 or Φ_1 value is computed from four appropriately selected bytes associated with a (i', j') pair of indices deduced from (i, j)).

It is easy to see that the σ key addition transformation just results in XORing the $\Phi_0[X, Y, Z, W]$ (resp $\Phi_1[X, Y, Z, W]$) value associated with a (X, Y, Z, W) quartet of bytes with a $\Phi_0[K_X, K_Y, K_Z, K_W]$ (resp $\Phi_1[K_X, K_Y, K_Z, K_W]$) 4-bit constant which depends on four subkey bytes.

We now investigate the effect of the S_0 and S_1 S-boxes of the γ non linear part of the Crypton round function on the $\Phi_0[X, Y, Z, W]$ (resp $\Phi_1[X, Y, Z, W]$) class values. For that purpose, for each of the S_0 and S_1 S-boxes, we compute the values

$$\begin{gathered}\#\{\text{X,Y,Z,W}\in[0,255]^4\ /\Phi_0\ [\text{X,Y,Z,T}]=\text{a} \\ \text{and } \Phi_0\ [S_\epsilon(\text{X}),S_\epsilon(\text{Y}),S_\epsilon(\text{Z}),S_\epsilon(\text{T})]=\text{b}\}\text{-}(256)^3 \\ \text{and} \\ \#\{\text{X,Y,Z,W}\in[0,255]^4\ /\Phi_1\ [\text{X,Y,Z,T}]=\text{a} \\ \text{and } \Phi_1\ [S_\epsilon(\text{X}),S_\epsilon(\text{Y}),S_\epsilon(\text{Z}),S_\epsilon(\text{T})]=\text{b}\}\text{-}(256)^3\end{gathered}$$

where ϵ takes values 0 or 1, and a and b are in [0,15].These values represent biases with respect to the average value $(256)^3$. We obtain four 16×16 matrices (one for Φ_0 and S_1 (see appendix A), one for Φ_0 and S_0, one for Φ_1 and S_1, one for Φ_1 and S_0). The columns of such matrices are indexed by a and their lines by b, and the value associated with column a and line b represents (up to a multiplicative factor of $(256)^4$) the bias of the transition probability from the class associated with the Φ input value a to the class associated with the Φ output value b.

Now it clearly results from the above properties of γ, $\tau \circ \pi$, and σ that the way one entire round of Crypton affects the Φ_0 or Φ_1 values can be represented by a 16×16 matrix of transition probabilities (or equivalently of biases) obtained by multiplying one of the four above matrices by the 16×16 key-dependent permutation matrix which entry associated with the a column and the b line is equal to 1 if $b = a \oplus \Phi_0[K_X, K_Y, K_Z, K_W]$.

Under the heuristic assumption that the behaviour of the cipher is nearly markovian, we can multiply those matrices to represent transitions of Crypton on n rounds. Of course the obtained matrix is key dependent (it depends upon 4 linear combinations of the key bits per round). However the orders of magnitude of the biases encountered in the product matrix are the same for most values of the keybits (they are larger than average for a few special values, e.g. when all 4-bit key words are equal to zero).

5.2 Attack Procedure

We present here a chosen plaintext of 8 inner rounds of Crypton (i.e. without the first key-addition and without the final transformation) which can be extended to an attack of the full 8-rounds version of Crypton (including the initial and the final transformation), with very similar performance. This basic attack is a 1-R attack based upon the above described computations of 6-rounds transition matrices (given by the product of 6 one-round transition matrices).

We select a (i, j) pair of first-round indexes and bit ϵ (0 or 1) and encrypt N chosen plaintext blocks such at the input to the first occurrence of the σ transformation (at the end of the first round), the $\Phi = \Phi(b, i, j)$ is equal to a constant 4-bit value α. We can expect the resulting Φ value associated with the input to the last round to be distributed according to unbalanced probabilities (or equivalently biases) given by the α column of the 6-rounds transition matrix.

We are using the χ^2 test in the same way as described in [HG97] to test four key bytes derived from the last round subkeys (i.e. those linear combinations of the last round subkey bits enabling to recover the four bytes involved in the Φ value of the input to the last round). For each of the 2^{32} key assumptions, we can partially decrypt the last round and compute (in at most 2^{32} operations, provided that ciphertext blocks have been first partitioned according to the value of a suitable 4-bytes word) the distribution of the Φ values at the input to the last round and the χ^2 indicator associated with the obtained distribution of 16 empiric frequencies. The obtained indicator is expected to be substantially higher for the right assumption on the four key bytes.

The N number of plaintexts required by the attack is inversely proportional to the sum of the squares of the a priori expected biases. Thus the required number of plaintexts can be deduced from the biases in the above introduced 6-round matrices. The best result are obtained with Φ_1 computations (instead of Φ_2 computations. For average values on the 6-uples of 4-bit key words, the obtained N value is close to 2^{112}. For the best 6-uple values (e.g. the null 6-uple), about 2^{104} suffice to recover the 32 last round key bits.

So in summary we obtain 32 bits of information about the last round keybit using 2^{112} chosen plaintexts. Therefore we can recover the entire last round subkey (and then, once having decrypted the last round, derive the other subkeys, using the same method) with say 2^{116} chosen plaintexts.

6 Results Concerning Crypton v1.0

Crypton v1.0 was introduced by Chae Hoon Lim in [Li99] to modify the initial key schedule and improve the S-boxes.

Instead of using two S-boxes like the initial version, Crypton v1.0 uses four S-boxes S_0, S_1, S_2, S_3 which verify $S_0^{-1} = S_2$ and $S_1^{-1} = S_3$. The γ_o and γ_e transformations are redefined as follows :

$$B = \gamma_o(A) \Leftrightarrow b_{i,j} = S_{i+j \bmod 4}(a_{i,j})$$
$$B = \gamma_e(A) \Leftrightarrow b_{i,j} = S_{i+j+2 \bmod 4}(a_{i,j})$$

We still have : $\gamma_o^{-1} = \gamma_e$ and $\gamma_e^{-1} = \gamma_o$ and thus the encryption and decryption processes are still identical.

The new S-boxes S_0, S_1, S_2, S_3 are all designed from an 8x8 involutive S-box S, chosen for its good diffusion properties. The purpose of the replacement of the Crypton S-boxes was to lower the number of the most probable low weight differential and linear characteristics, and thus to speed up the diffusion achieved by Crypton.

According to our computer experiments, these S-box modifications very significantly improve the resistance of Crypton against the stochastic attacks presented in Sections 4 and 5.

Let us first consider stochastic cryptanalysis using differential properties on 6 rounds of Crypton v1.0. We found that if input difference is the (49, 49) pair of Δ_1 values, then the probability that the output be a (δ_1, δ_2) pair with $(\delta_1, \delta_2) \in D_1 \times D_1$ is $2^{-151.23}$, taking into account all intermediate values with an alternation of elements of Δ_1 and Δ_2. The $2^{-151,23}$ value represents the best probability associated with 6 rounds of Crypton v1.0. It is significantly lower than the $2^{-112.62}$ best probability associated with 6 rounds of Crypton.

For stochastic cryptanalysis using a partition of blocks in 16 classes, the number of plaintexts required for a 1-R attack is at least 2^{135}. This figure was derived from Φ_1, and corresponds to the most favorable values of the 6-uple of 4-bit key words involved in the computations. It is significantly higher than the $N = 2^{104}$ minimal number of required plaintexts obtained for the initial Crypton, and higher than the number of distinct plaintexts (2^{128}).

Thus in summary Crypton v1.0 resists the stochastic attacks of Sections 4 and 5 much better than the initial version of Crypton. This seems to be a direct consequence of the design criteria of the new Crypton S-boxes. Because of the decrease of the number of low weight highest probability differential and linear characteristics at each round, the number of "high probability paths" that together form the transition probabilities considered in our stochastic attacks is decreased and the performance of the attacks is significantly affected. Changes in S-boxes proposed in Crypton v1.0 were very discerning.

7 Conclusion

We have described two stochastic attacks on the eight-round version of the Crypton block cipher which are faster than an exhaustive search and more efficient than the best attacks discovered so far.

The first of these two attacks is close to a truncated differential attack, but we believe that probability matrices computations are more precise than truncated differentials probabilities computations would be. It seems to us that an analysis based on truncated-differential probabilities would have led to an overestimate of the performance of the attack.

Our attacks do not threaten the security of Crypton in a full version, but nevertheless put the highlight on a (slight) diffusion weakness in the linear part of the Crypton round function. Even if finding the most relevant cryptographic criteria on the linear part of substitution-permutation blockciphers is still to a large extent an open issue, diffusion criteria related to the number of "active" S-boxes (e.g. MDS properties) offer some clues. Our attacks exploit the existence of low weigh (and iterative) relations between the input and the output of the linear part of Crypton, which lead to statistics involving only 4 active S-boxes per round.

Finally, the comparison between the results obtained on the initial Crypton and Crypton v1.0 confirms that the S-box modifications introduced in Crypton v1.0 significantly improve its resistance against some classes of attacks.

References

Ba99. O. Baudron, H. Gilbert, L. Granboulan, H. Handschuh, A. Joux, P. Nguyen, F. Noilhan, D. Pointcheval, T. Pornin, G. Poupard, J. Stern, S. Vaudenay, "Report on the AES Candidates", *The Second Advanced Encryption Standard Candidate Conference,* N.I.S.T., 1999.

Hal99. C. D'Halluin, G. Bijnens, V. Rijmen, B. Preneel, "Attack on Six Rounds of Crypton". In *Fast Software Encryption - FSE'99* , p. 46, Springer Verlag, Rome, Italy, March 1999.

HG97. H. Handschuh, H. Gilbert, "χ^2 Cryptanalysis of SEAL Encryption Algorithm". In *Fast Software Encryption - FSE'97*, pp. 1-12, Springer Verlag, Haifa, Israel, 1997.

Li98. C.H. Lim, "Crypton : A New 128-bit Block Cipher", *The First Advanced Encryption Standard Candidate Conference,* N.I.S.T., 1998.

Li99. C.H. Lim, "A Revisited Version of Crypton : Crypton V1.0". In *Fast Software Encryption - FSE'99* , p. 31, Springer Verlag, Rome, Italy, March 1999.

Ma93. M. Matsui, "Linear Cryptanalysis Method for DES Cipher". In *Advances in Cryptology - Eurocrypt'93*, pp. 386-396, Springer Verlag, Lofthus, Norway, 1993.

LM91. X. Lai, J.L.Massey and S. Murphy, "Markov Ciphers and Differential Cryptanalysis". In *Advances in Cryptology - Eurocrypt'91*, p. 17, Springer Verlag, Brighton, UK, 1991.

HM97. C. Harpes and J.L.Massey, "Partitioning Cryptanalysis". In *Fast Software Encryption - FSE'97* , p. 13, Springer Verlag, Haifa, Israel, January 1997.

Mu. S. Murphy, F. Piper, M. Walker, P. Wild, “Likehood Estimation for Block Cipher Keys”. Unpublished.

Va95. S. Vaudenay, “La sécurité des Primitives Cryptographiques”. Doctoral Dissertation, 1995.

A Matrix of Transition Biases for the Φ Value

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	204800	-36864	124928	-28672	-53248	77824	-18432	69632	4096	-77824	-30720	-86016	-114688	36864	-116736	45056
1	-28672	-83968	-4096	-94208	-94208	63488	-118784	36864	143360	-47104	151552	-36864	-20480	100352	-28672	61440
2	-55296	53248	-28672	45056	112640	-94208	53248	-86016	-75776	94208	-69632	102400	51200	-53248	12288	-61440
3	-69632	102400	-94208	75776	28672	-45056	53248	-51200	-45056	61440	-53248	22528	86016	-86016	94208	-79872
4	-114688	53248	-112640	45056	-36864	-94208	6144	-86016	4096	94208	-14336	102400	106496	-53248	161792	-61440
5	-20480	67584	-12288	110592	143360	-47104	135168	-53248	-126976	30720	-102400	53248	4096	-83968	-20480	-77824
6	34816	-36864	77824	-28672	-92160	77824	-102400	69632	63488	-77824	118784	-86016	-38912	36864	-61440	45056
7	86016	-86016	77824	-92160	-45056	28672	-36864	67584	61440	-45056	36864	-38912	-102400	69632	-77824	96256
8	28672	-135168	14336	-126976	-57344	12288	-71680	4096	90112	-12288	88064	-20480	-36864	135168	-55296	143360
9	102400	-2048	94208	-12288	-61440	145408	-53248	118784	45056	-129024	20480	-118784	-86016	18432	-61440	-20480
10	-55296	36864	-61440	28672	112640	-77824	86016	-69632	-75776	77824	-102400	86016	51200	-36864	45056	-45056
11	-86016	102400	-77824	75776	45056	-45056	36864	-51200	-61440	61440	-36864	22528	102400	-86016	77824	-79872
12	-77824	118784	-59392	110592	106496	4096	116736	12288	-73728	-4096	-75776	4096	20480	-118784	43008	-126976
13	-53248	-14336	-77824	28672	12288	-129024	36864	-135168	-61440	112640	-69632	135168	102400	-2048	110592	4096
14	34816	-53248	45056	-45056	-92160	94208	-69632	86016	63488	-94208	86016	-102400	-38912	53248	-28672	61440
15	69632	-86016	94208	-92160	-28672	28672	-53248	67584	45056	-45056	53248	-38912	-86016	69632	-94208	96256

Table 1. Matrix of distribution for S_1 and Φ_0

Bitslice Ciphers and Power Analysis Attacks

Joan Daemen, Michael Peeters, and Gilles Van Assche

Proton World Intl.
Rue Du Planeur 10, B-1130 Brussel, Belgium
Email: {daemen.j, peeters.m, vanassche.g}@protonworld.com
http://www.protonworld.com/research

Abstract. In this paper, we present techniques to protect *bitslice* block ciphers against power analysis attacks. We analyze and extend a technique proposed in [12]. We apply the technique to BaseKing, a variant of 3-Way[9] that was published in [7]. We introduce an alternative method to protect against power analysis specific for BaseKing. Finally, we discuss the applicability of the methods to the other known bitslice ciphers 3-Way and Serpent [1].

1 Introduction

The inherent security offered by a block cipher is best evaluated by investigating the cipher's resistance against the set of known cryptanalytic attacks. It is generally agreed that a cipher for which there are attacks that are more efficient than exhaustive key search have an inherent weakness. In our opinion, the absence of this kind of attacks is rightfully the primary criterion for comparison of ciphers.

Although interesting, in many cases the relevance of these cryptanalytic attacks is mostly academic: in practical applications of the cipher they turn out to be irrelevant for several reasons. The attacks might require an unrealistic amount of plaintext/ciphertext pairs, such as differential cryptanalysis and linear cryptanalysis of DES [2,15]. In other cases, the theoretical weakness only manifests itself in very rare cases. This is the case for weak keys such as in IDEA [10]. In still other cases there is an easy way to protect against the weaknesses. Weak keys of DES and related-key attacks [11] can be avoided by generating keys independently or by deriving them using one-way functions.

In the model for theoretical cryptanalysis, the key is considered to be unknown and an attacker has access only to plaintext and ciphertext, and can possibly manipulate the key in certain ways. He has no access to intermediate computation results. In practical implementations the secrecy of the key, that is considered to be a given in cryptanalysis, must be accomplished by effective physical and logical protection. This is invariably the most expensive and problematic aspect of any serious application using cryptography. Particularly challenging are distributed applications, where smart cards used by consumers, and terminals used by merchants and service providers, use cryptography to secure fund transfers or the conditional access to services (e.g., GSM). As has been shown recently in several publications [14,5], implementing cryptographic algorithms on these platforms is not trivial. One of the reasons for these problems is that a combination of hardware characteristics and algorithm coding or the presence of (induced) errors might give away information on intermediate computation results. By using statistical techniques, this information is then used to find the key.

In the long term, this problem can best be dealt with by incorporating on the smartcards and terminal security modules dedicated cryptographic hardware that performs integrity checks and minimises the leakage of information. Such components are not available yet and use must be made from existing components. In this paper, we discuss techniques to protect implementations of bitslice ciphers on state-of-the-art smartcards against power analysis attacks.

2 Implementation Attacks

2.1 Timing Attacks and Simple Power Analysis

In timing attacks, the dependence of the execution time of the cipher on plaintext or key bits is exploited to derive key or plaintext information. An effective protection against timing attacks is writing the code in such a way that the number of cycles taken by an execution is independent of the value of key or plaintext bits (preferably a constant).

In so-called *simple power analysis* attacks, the attacker makes use of some quantity, measurable at the outside of a cryptographic device, to detect instructions being executed inside it. Typically, this measurable quantity is the power consumption or radiation. This may leak information on key or plaintext bits if the instructions that are executed depend on the values of data that are being processed. An effective way to protect against this type of information leak is to program the cipher as a fixed sequence of instructions. This also implies a constant execution time, effectively sealing up the timing leak.

2.2 Differential Power Analysis

In more advanced power analysis attacks, e.g., *differential power analysis* (DPA) [14] the correlation between the power consumption (or radiation, ...) and the values of the operands of the instructions is exploited. Usually, this correlation is weak with respect to the noise on the power consumption. Even if no special measures are taken, several (tens to thousands, depending on the type of instruction and quality of the processor) cipher executions are required to exploit these correlations in an attack.

The basic principle of differential power analysis is that the probability distribution of the power consumption, given that a certain event occurs, can be distinguished from the average probability distribution. The attack is mounted as follows:

- The first step of the attack is to instruct the cryptographic device to perform a number of cipher computations. For each of these computations the power consumption pattern P_i is measured and stored together with the known parameters of the computation, that is, a_i the plaintext or the ciphertext. This is the data acquisition phase which produces a data set $D = \{(a_i, P_i) \mid i = 1 \ldots z\}$.
- Then, an event is specified whose occurrence depends on the value of a number of plaintext (or ciphertext) bits and key bits we wish to determine. Such an event can be, for instance, that the result of an intermediate cipher calculation gives a certain result (which must be at some time present in a CPU register, ALU, bus, or memory cell). We call the *target subkey* the key bits the specified event depends on.
- For each of the possible values of the target subkey, the following check is performed. In the hypothesis that the target subkey s^* is correct, the set of power consumption patterns are divided into two groups: those for which the event occurs $D_1 = \{(a_i, P_i) \mid f(s^*, a_i) = 1\}$ and the complementary set $D_0 = \{(a_i, P_i) \mid f(s^*, a_i) = 0\}$. (As suggested by these formulas, f indicates whether the event occurs given the known and hypothesis values.)

  It is assumed that the two subsets D_1 and D_0 can be statistically distinguished for the correct hypothesis. We therefore define some distance between the two distributions. The subkey value for which this distance is maximized, is taken as the correct value. In general, a wrong target subkey value will divide the power consumption patterns in two sets in which the event occurs an average number of times. (However, if the round function has certain algebraic properties, several subkey values, among which the correct one, may be suggested.)

2.3 Higher-Order DPA

Depending on the usage of the power consumption pattern we can distinguish differential power attacks of different orders. Generally speaking, N-th order DPA makes use of N different intermediate values calculated at N different times during the execution of the cipher algorithm.

- In first-order DPA as described by Kocher [14], the event mentioned above is typically the fact that a particular bit (or set of bits) in a CPU register, bus, or memory has the value 1 (or 0). It is usually sufficient to distinguish the two data sets D_0 and D_1 by their average.
- In second-order DPA, the event is typically the fact that two bits of operands occurring at different times during the computation, are equal (or different). This situation occurs when one can group a set of samples according to the value of the exor of two operand bits rather than an absolute value of an operand bit.

Protection against DPA can take different forms. Here are two examples:

– The alignment of the power consumption patterns can be made harder by building in random time differences in the cipher execution. This reduces the effectiveness of the attack described above by requiring more power consumption patterns.
– Theoretically, first-order DPA can be made impossible by programming the cipher in such a way that the operands bear no correlation with intermediate block cipher states or key bits. The techniques proposed in this paper attempt to do exactly that, through the introduction of random biases and state/key splitting. This can be generalized to resistance against N-th order DPA, where it is required that no set of N operands has a correlation with state or key. These techniques were already proposed in [4].

In practice, a second-order attack is more difficult to mount than a first-order one. The reasons for this are:

– **A more complex layout.** In first-order DPA, the probability distributions are one-dimensional, e.g., they represent the power consumption at a given stage in the cipher computation. Usually, for any given stage a subkey hypothesis can be tested by taking the average for the two subsets and use the difference between them as distance. In the second order DPA, the equality (or inequality) of two bits is of interest. Here, a subkey hypothesis has to be tested by determining whether it divides the power samples in two groups with the following properties: in one group, the two power consumption samples have a tendency to increase or decrease together, in the other group they fluctuate in opposite directions. Computing the distance between the two distributions takes more computations than just taking the difference between the averages. More complex processing is thus required.
– **Increased memory and processing requirements.** Especially when a cipher implementation uses random delays, the exact location of the cycles where a certain operand is processed in the power consumption samples is unknown a priori. Hypothesis testing has to be performed for all possible locations. If n samples are of interest for a first order attack, one gets n^2 pairs of samples for a second order attack, thus greatly increasing the demand of data storage and processing.
– **Increased number of power consumption patterns.** To distinguish two distributions from each other, one needs enough samples before a statistically significant result appears. For the same amount of noise, bi-dimensional distributions are harder to distinguish than their equivalent 1D distributions. This is detailed in appendix A. Typically, if z power consumption patterns are needed in the 1D case, about $2z^2$ patterns are necessary in the 2D case. Furthermore, the use of random delay spreads the effect of the event in a single dimension, decreasing the signal-to-noise ratio linearly. In second-order DPA, this effect is spread in two dimensions, decreasing the signal-to-noise ratio quadratically.

Before discussing the different protection methods, we discuss correlation and decorrelation. For the sake of brievety, no proofs are given in this version of the paper.

2.4 On Correlation and Decorrelation

The correlation between two binary variables (bits) f and g is given by $C(f,g) = 2\Pr(f = g) - 1$. If f and g can be expressed in terms of a word a of n bits $(a_1, a_2, \ldots, a_n)$, i.e., $f(a), g(a)$, this can be computed as [7,8]:

$$C(x,y) = 2^{-n} \sum_a (-1)^{f(a) \oplus g(a)}.$$

Consider a bit f that can be expressed as a function of two words a and b, i.e., $f(a,b)$. This bit is said to be decorrelated from a if for any linear combination of bits of a, denoted by $u^{\mathrm{t}}a$ (with u a *selection vector* [8]), we have $C(f, u^{\mathrm{t}}a) = 0$.

A word d that is a function of two words a and b is said to be decorrelated from a if for all linear combinations of bits of d, denoted by $w^{\mathrm{t}}d$, and all linear combinations of bits of a, denoted by $u^{\mathrm{t}}a$, we have $C(w^{\mathrm{t}}d, u^{\mathrm{t}}a) = 0$. Clearly, this implies that all bits of d are also decorrelated from a.

In words, knowledge of a bit f (or word d) gives no information whatsoever on the word a if the word b is unknown.

Consider an operand $a'(a, \delta)$ that is defined by $a' = a$ if $\delta = 0$ and $a' = \bar{a}$ otherwise. Using techniques introduced in [8] it can easily be verified that all bits of a' are decorrelated from a. The complete word a' is however not decorrelated from a. δ is called the *masking bit.*

For an operand $a'(a, a'')$ that is defined by $a' = a \oplus a''$ it can easily be shown that it is decorrelated from a. Obviously, thanks to symmetry, the operand $a'' = a \oplus a'$ is also decorrelated from a. a'' is called the *masking word.*

It can be shown that a word b that is the result of the computation of two operands $a'_1(a_1, a''_1)$ and $a'_2(a_2, a''_2)$ is decorrelated from both a_1 and a_2 if a''_1 and a''_2 are mutually decorrelated (or equivalently, independent).

Moreover, all bits of a word b that is the result of a bitwise logical computation of two operands $a'_1 = a_1 \oplus a''_1$ and $a'_2 = a_2 \oplus a''_2$ are decorrelated from both a_1 and a_2 if the bits at corresponding positions of masking variables a''_1 and a''_2 are mutually decorrelated (or equivalently, independent).

Finally, for a state $a = a' \oplus a''$, any computation involving only terms of a' (or only a''), will have only operands that are decorrelated from a.

3 The Duplication Method

Louis Goubin and Jacques Patarin [12] propose the *Duplication Method*: a method for increasing DPA-resistance based on secret sharing. It aims to remove all correlation between operands and intermediate state or key, thus making first-order DPA impossible. In this section, we discuss the limitations of the methods as proposed in [12].

The basic principle of the Duplication Method is to split any variable V of a given cipher computation into a set of k other variables $V_1, V_2, \dots V_k$ such that the variable can be reconstructed through the use of a function f: $V = f(V_1, \dots V_k)$. The cipher computations are performed on the variables V_i such that the relation $V = f(V_1, \dots V_k)$ holds at all times and without having to calculate V explicitly. Furthermore, the knowledge of $k-1$ of the variables V_i does not give any information on V itself.

In the practical methods they propose, the Duplication Method is applied to DES and the variables V are split in two parts V_1 and V_2 with $V = V_1 \oplus V_2$. For the linear operations in DES (expansion, bit permutation, exor of output of f-function), the computation can be done on V_1 and V_2 separately and independently (for a linear function L, we have $L(V_1) \oplus L(V_2) = L(V_1 \oplus V_2) = L(V)$). For the key addition it is sufficient to add it to one of the two variables.

For the S-box evaluation however, the computations involving V_1 and V_2 cannot be kept separated due to the nonlinearity of the S-boxes. Goubin and Patarin [12] propose the use of 12-bit to 8-bit lookup tables $T_i(v_1, v_2)$ satisfying

$$(v_1', v_2') = T(v_1, v_2) = (A(v_1, v_2), S(v_1 \oplus v_2) \oplus A(v_1, v_2)) \tag{1}$$

where A is a *randomly-chosen* secret transformation and S is the original 6-bit or 4-bit S-box. Clearly, $v_1' \oplus v_2' = S(v_1 \oplus v_2)$, so that the table-lookup preserves the condition $v = v_1 \oplus v_2$. The computation of v_1', v_2' is performed as a lookup in a table of 4096 bytes. Unfortunately, the size of these tables that replace the S-boxes makes this method prohibitively expensive for current smartcards, where memory is a scarce resource.

For this reason, they propose a variant of their method that uses more compact lookup tables. The S-box computation is performed in the following two steps:

$$v_0 = \varphi(v_1 \oplus v_2), \tag{2}$$

and

$$(v_1', v_2') = S'(v_1, v_2) = (A(v_0), S(\varphi^{-1}(v_0)) \oplus A(v_0)) \tag{3}$$

with φ a secret bijective function.

Although φ can be chosen such that v_0 can be calculated by combining individual computations (e.g., if φ is linear, it reads $v_0 = \varphi(v_1) \oplus \varphi(v_2)$), the intermediate state value v is fully determined by a single operand: v_0. The inability to exploit v_0 for hypothesis testing is only based on on the secrecy of φ. Apart from disclosure by a manufacturer, an attacker can learn more about φ if he or she has access to a sample card where the cipher can be run with a known key. The problem can thus be factored in two sub-problems, namely, learning more about φ and mounting a first-order DPA attack focusing on the v_0 values rather than v. Moreover, the linearity of φ (or the fact that it is quadratic) may really help a lot in determining it.

From an academic point of view, both variants of the method face the problem that they do not guarantee decorrelation between operands and intermediate state values.

For the first variant of the Duplication Method, the lookup-table output (v_1', v_2') can be expressed in terms of (v_1, v) using an equivalent table T' by including the linear transformation $(v_1, v_2) = (v_1, v \oplus v_1)$. To have decorrelation of the bits of v_1', v_2' from v, it is a requirement that $C(b, u^{\mathrm{t}} v) = 0$ for all 8 bits b of v_1' or v_2' and for all 64 possible selections u. If A is chosen randomly it is very unlikely that this is the case.

For the second variant of the Duplication Method, bits of operand v_0 are correlated to linear combinations of bits of v. As a matter of fact, we have:

$$\sum_u C^2(b, u^{\mathrm{t}} v) = 1$$

for any of the output bits. In the case that φ is linear (as proposed in [12]), every bit of v_0 is correlated to a linear combination of bits of v with correlation 1.

4 Bitslice Ciphers

Bitslice ciphers can be implemented using only bitwise logical instructions and (cyclic) shifts. The term bitslice cipher was introduced by Eli Biham referring to the AES candidate Serpent [1] designed by Eli Biham, Ross Anderson and Lars Knudsen. Older examples of bitslice ciphers are 3-Way[9] published in 1993 and BaseKing. BaseKing is a variant of 3-Way that was described in [7] but never presented at a conference.

4.1 BaseKing

BaseKing has a block and key length of 192 bits (24 bytes). It is an iterated block cipher with a round transformation composed of a number of steps, each with its own function. These steps treat the intermediate encryption result, called the *state*, in a *uniform* way. The state, denoted by a consists of 12 16-bit words denoted by a_0 to a_{11}. The round transformation has 5 steps:

- **key addition:** the cipher key and a round constant is added to the state.

$$a \leftarrow a \oplus k \oplus Cr_j$$

- **diffusion:** the words are transformed with a linear transformation with high diffusion (branch number 8):

$$a_i \leftarrow a_i \oplus a_{i+2} \oplus a_{i+6} \oplus a_{i+7} \oplus a_{i+9} \oplus a_{i+10} \oplus a_{i+11}$$

- **early shift:** the words are cyclically shifted over 12 different offsets:

$$a_i \leftarrow a_i \lll r_i$$

– **S-box:** the words are transformed with a nonlinear transformation operating in parallel on sets of 3 bits:

$$a_i \leftarrow a_i \oplus (a_{i+4} \vee \overline{a_{i+8}})$$

– **late shift:** the words are cyclically shifted over 12 different offsets:

$$a_i \leftarrow a_i \gg r_{11-i}$$

The vector of rotation constants used in the shift operations is

$$r = (0, 8, 1, 15, 5, 10, 7, 6, 13, 14, 2, 3).$$

The round constants are given by:

$$Cr_j = (0, 0, q_j, q_j, 0, 0, 0, 0, q_j, q_j, 0, 0)$$

with q_j given by the following pseudo-c program:

```
q[0] = 0x000B;
if ((q[j+1] = q[j]<<1) & 0x0100) q[j+1]^= 0x0111;
```

BaseKing has 11 rounds and a final output transformation. The final output transformation consists of a key addition and a diffusion step (as described above) followed by a transformation that inverts the order of the words:

$$a_i \leftarrow a_{11-i}$$

Thanks to the arrangement of the steps and the algebraic properties of the operations, the inverse cipher is exactly the same as the cipher itself, with the exception of the round constants. For a detailed treatment of these aspects, we refer to [7].

4.2 Cryptanalysis

The design of BaseKing aims at providing strong resistance against differential and linear cryptanalysis and the absence of symmetry properties. We refer to [7] for a development of this point.

For 3-Way, it has been shown that the lack of a real key schedule allows mounting of a related-key attack [13]. This attack is also applicable to the cipher BaseKing. However, in applications requiring only encryption, MACing and key derivation, related-key attacks can be easily prevented by the application of sound key management principles, i.e., by avoiding key variants.

5 Protecting Bitslice Ciphers against DPA

This section describes our methods of protecting bitslice ciphers against first order DPA attacks.

Before executing the cipher, the initial value of the state a, i.e., the plaintext, is split into two state shares a' and a'' with $a = a' \oplus a''$. All computations will be performed on the state shares in such a way that the relation $a = a' \oplus a''$ holds at all times. The linear steps (early shift, diffusion, late shift) can be applied to the state shares a' and a'' independently and therefore provide decorrelation from state words.

5.1 Key Addition

The key addition can be applied by adding the round key to one of the two split states: $a' \leftarrow a' \oplus k$. In the assumption that the attacker has no information on a', a first-order DPA cannot be used to gain information on the key k.

However, in [3], Eli Biham and Adi Shamir describe an attack that uses Hamming weight information on round key words to retrieve the key. The Hamming weight information is obtained by taking the average consumption over multiple cipher computations with the same key. The ability to use this Hamming weight information to derive the key strongly depends on the key schedule. For DES (or Triple-DES) the complete key can be found using "standard techniques from error correcting codes".

Ironically, its lack of a key schedule gives BaseKing an excellent protection against this attack. The cipher key is applied at the end of every round by just exoring it with the state. The subkey words are the same for every round. In the case of 8-bit words, this attack gives on the average 2.54 bits of information per byte, leaving still $24 * (8 - 2.54) = 131$ bits to guess. In the case of 32-bit words, this becomes 3.55 bits per word, leaving still $6 * (32 - 3.55) = 171$ bits to guess.

Anyway, the knowledge of key information might be exploited to further attack the cipher. A simple way to protect against the key schedule attack is to apply secret sharing on the key. The key k is split into two parts k' and k''. The exor with k is then executed by a word-by-word exor of the state with k' followed by a word-by-word exor of the state with k''. The addition of two subkeys per round can even be done with some linear steps in between, if one of the two subkeys undergoes a linear transformation: $L(a + k) = L(a) + k_1$ with $k_1 = L(k)$.

In combination with the state secret sharing method, the key secret sharing method can make key addition really symmetric: $a' \leftarrow a' \oplus k'$ and $a'' \leftarrow a'' \oplus k''$.

5.2 Full State Splitting

Similar to the duplication method of Goubin and Patarin, the state a is split in a' and a'' with a' generated randomly before the computation and only recombined at the end of the cipher computation.

The BaseKing S-box operates on sets of three words of the state (e.g., a_0, a_4 and a_8), and transforms them by

$$a_i \leftarrow a_i \oplus (a_{i+4} \oplus 1)a_{i+8} \oplus 1, \tag{4}$$

with index additions modulo 12. Applying secret sharing gives rise to $a_i = a'_i \oplus a''_i$, $i = 0, 1, \ldots 11$. We must determine functions f' and f'':

$$a'_i \leftarrow f'(a'_i, a'_{i+4}, a'_{i+8}, a''_i, a''_{i+4}, a''_{i+8}) \quad \text{and} \tag{5}$$

$$a''_i \leftarrow f''(a'_i, a'_{i+4}, a'_{i+8}, a''_i, a''_{i+4}, a''_{i+8}) \tag{6}$$

that preserve the relation $a = a' \oplus a''$:

$$f' \oplus f'' = a'_i \oplus a''_i \oplus (a'_{i+4} \oplus a''_{i+4} \oplus 1)(a'_{i+8} \oplus a''_{i+8}) \oplus 1, \quad i = 0, 1, \ldots 11. \tag{7}$$

The restriction on f' and f'' is that during their computation there are no operands that bear correlation with a. Using the distribution rule of $\mathbf{F}_2$, we get:

$$\begin{aligned} f' \oplus f'' = {} & a'_i \oplus a''_i \oplus a'_{i+8} \oplus a''_{i+8} \\ & \oplus a'_{i+4}a'_{i+8} \oplus a'_{i+4}a''_{i+8} \oplus a''_{i+4}a'_{i+8} \oplus a''_{i+4}a''_{i+8} \oplus 1. \end{aligned} \tag{8}$$

A computation involving only components of a' or a'' cannot involve operands that have a correlation with a. However, due to the presence of the mixed terms, i.e., with components of a' and a'', the computation of f' will necessarily involve terms of a'' or vice versa. For instance, one gets:

$$\begin{aligned} f' &= a'_i \oplus a'_{i+8} \oplus a'_{i+4}a'_{i+8} \oplus a'_{i+4}a''_{i+8} \\ f'' &= a''_i \oplus a''_{i+8} \oplus a''_{i+4}a'_{i+8} \oplus a''_{i+4}a''_{i+8} \oplus 1. \end{aligned}$$

To guarantee decorrelation of all operands, the order in which these functions are computed is important. Consider the expression for f' given above. If it is evaluated from right to left, after the addition of the two rightmost terms, the following operand occurs: $a'_{i+4}a''_{i+8} \oplus a'_{i+4}a'_{i+8} = a'_{i+4}a_{i+8}$. Clearly each bit of this operand has a correlation of 1/2 with the corresponding bit of state word a_{i+8}. Since a' is random and independent of a, we have:

$$a'_{i+4}a_{i+8} = \begin{cases} a_{i+8} & \text{when } a'_{i+4} = 1, \\ 0 & \text{when } a'_{i+4} = 0, \text{ and thus} \end{cases} \tag{9}$$

$$C\left(a'_{i+4}a_{i+8}, a_{i+8}\right) = \frac{1}{2}. \tag{10}$$

If the expression for f' is evaluated from left to right, it can be shown that no operands occur that have a correlation with the state. The computations of all terms except the last one involve only words of a', hence here decorrelation from a is automatic. For the addition of the last term to the intermediate result of the computation, the presence of a'_i in the first term implies that the masking words of two terms are decorrelated.

The state splitting method can be generalized to provide protection against second and higher order DPA attacks. In principle, this enables the smartcard designer to adjust the level of security by making DPA attacks arbitrarily difficult. For instance, protecting against second-order DPA requires the state a to be split into three parts, namely $a = a' \oplus a'' \oplus a'''$. The evaluation of the S-box requires care when deciding in which order the operations must be performed.

5.3 The Bias Vector Method

In the bias vector method one of the two split states is in a particular sub-class that can be kept invariant under the cipher computations. For BaseKing this is the class of states where each 16-bit word is either all-0 (0) or all-1 ($\bar{0}$). These particular split states are called *bias states.* A bias state can be represented by a 12-bit vector, called a *bias vector.*

A bias state is invariant under the shift operations. The diffusion operation maps every bias state to another bias state, since it operates in parallel on the bits of the words. Thanks to the linearity of the diffusion operation, computing the bias vector at the output of a diffusion step from the bias vector at its input can be done with some table-lookups and exors. For example: 3 table lookups in tables with 16 (2^4) entries and two exors. Thanks to their compactness, input-output bias vector pairs can be computed beforehand and stored in memory for later usage.

The cipher computation operates on a biased state A equal to $a \oplus d$. The bias vector corresponding with d is denoted by δ.

For the linear steps including the key addition, the computation is performed on A. The evolution of the bias vector in the linear steps is computed using the table-lookups described above. For each round a new random 12-bit bias vector is introduced.

We explain the computation of the first word of the output of the nonlinear transformation corresponding with:

$$a'_0 = a_0 \oplus (a_4 \vee \overline{a_8})$$

All other words are computed in the same way. The computation of A'_0 is done as follows:

1. **Computation of required values:** Compute $A_4 \oplus A_8$ and store it in a register. Store 0 in a register (A_4 and A_8 are already assumed to be in registers);
2. **Nonlinear computation:** We compute $A_4 \vee \overline{A_8}$ and store it as G.
3. **Computation of correction term**: depending on the bias vector bits δ_4, δ_8, one of the four registers containing 0, A_4, A_8 or $A_4 \oplus A_8$ is selected and its complement is stored. The selected register or the one containing its complement (depending on γ) is exored to G:
   – $\delta_4 = 0, \delta_8 = 0$: complement 0 and store as H. If $\gamma = 0$ add 0 to G, else add H to G;
   – $\delta_4 = 0, \delta_8 = 1$: complement A_4 and store as H. If $\gamma = 0$ add H to G, else add A_4 to G;
   – $\delta_4 = 1, \delta_8 = 0$: complement A_8 and store as H. If $\gamma = 0$ add H to G, else add A_8 to G;
   – $\delta_4 = 1, \delta_8 = 1$: complement $A_4 \oplus A_8$ and store as H. If $\gamma = 0$ add $A_4 \oplus A_8$ to G, else add H to G;

   This is programmed such that the sequence of instructions is independent of the branch that is taken, the only difference are the source/target registers.

4. **Computation of A'_0 and δ'_0:** $A'_0 = A_0 \oplus G$ and $\delta'_0 = \delta_0 \oplus \gamma$.

This computation can be repeated for all state words. The bits of all operands in the computation are decorrelated from state words:

- Bits of A_4 and A_8 are decorrelated independently (via d_4 and d_8) from state words, hence the results of $A_4 \oplus A_8$ and $A_4 \vee \overline{A_8}$ are also decorrelated.
- Depending on the values of δ_4 and δ_8 the operand is $0, A_4, A_8$ or $A_4 \oplus A_8$. The bits of individual operands are all decorrelated, except 0 that is obviously constant.
- In the subsequent computations of G and A'_0 the bits of all operands are decorrelated.

The advantage of the bias vector method is that second order DPA will be more difficult to accomplish than in the case of full state splitting thanks to the compact (and possible delocalized) processing of the bias vector. Unfortunately, the bias vector method has the disadvantage that decorrelation is only reached at bit-level and not at word level (e.g., if in a word a_i of the state the two LSB bits are equal, the two LSB bits of A_i will be equal). In cyclic shift operations, this might give away information via second-order effects.

5.4 Coding Results

To evaluate the implementability and cost of the methods described above, several versions of BaseKing were programmed on the ARM7 RISC processor (see `www.arm.com` for technical and other information). This processor is well know for its very high computation power vs. consumption ratio, and hence is ideal for smartcards. Special care has been taken to guarantee immunity against timing (for instance, handling of pipeline clearing) and SPA attacks. The following table summarizes the code size and execution time for the different versions.

version	cycles	code size
timing and SPA resistant	1949	776
full state splitting	4593	1148
bias vector	4505	2804
bias vector, guarded instr.	3845	1844

Clearly, the application of the anti-DPA methods has a considerable cost in execution time and code size. In the case of the bias vector method, most of the overhead comes from the computation of the S-box (2119 cycles, 1216 bytes code size).

We did not conduct actual power measurements and therefore were unable to verify whether the use of guarded instructions endangers the immunity against DPA attacks, and hence the last line of the table is only included for information.

More detailed information on the coding and updates will be made available at `http://www.protonworld.com/research`.

5.5 Applicability to 3-Way and Serpent

The bias vector method makes use of particular symmetry properties of the cipher BaseKing and cannot be extended to 3-Way or Serpent. The full state splitting method however can be extended to any bitslice cipher. For 3-Way, both the linear steps and the S-box evaluation can be done in exactly the same way as described for BaseKing.

In Serpent, the linear steps of the round function pose no problem. However, we do not expect that implementing the method for the Serpent S-boxes will be trivial. The BaseKing S-box mapping is very simple, containing only a single nonlinear term with only two factors per output word computation. Moreover, the expression is the same for all output words, only the input words differ. In Serpent, there are 8 different S-boxes, and the expressions of the output bits contain more terms with degrees up to 3. This is likely to give more mixed terms in the expressions of Serpent's equivalents of the f' and f'' functions and a relatively more important reduction in performance. Special care must be taken in the order of evaluation of these functions to guarantee correlation immunity (if possible). Moreover, due to the lack of symmetry in the Serpent S-boxes, this may give rise to an important overhead in code size.

Acknowledgments. We would like to thank Mr. Philip Theunissen for proofreading the final version of this paper.

6 Conclusions

We have applied and extended techniques to protect block ciphers against power analysis attacks to the bitslice block cipher BaseKing. These techniques have been validated by actual coding in assembly language on an ARM processor.

One of the techniques described generalises readily to the block cipher 3-Way. We have shown that this technique can be applied to Serpent, but more analysis and research is required to see what is the performance penalty in this case.

Finally, in the appendix we show the difference in power between first-order and second-order DPA with information-theoretical arguments.

References

1. E. Biham, R. Anderson, and L. Knudsen. Aes proposal serpent. *AES CD-1: documentation*, 1998.
2. E. Biham and A. Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.
3. E. Biham and A. Shamir. Power analysis of the key scheduling of the aes candidates. In *2nd AES Candidates Conference*, March 1999.
4. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. A cautionary note regarding evaluation of aes candidates on smart-cards. In *Proceedings of the 2nd AES Candidates Conference*, March 1999.

5. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology - CRYPTO'99*, pages 398–412. Springer-Verlag, 1999.
6. T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley & Sons, 1991.
7. J. Daemen. *Cipher and Hash Function Design.* PhD thesis, Katholieke Universiteit Leuven, March 1995.
8. J. Daemen, R. Govaerts, and J. Vandewalle. Correlation matrices. In R. Anderson, editor, *Fast Software Encryption*, pages 275–285. Springer-Verlag, 1994.
9. J. Daemen, R. Govaerts, and J. Vandewalle. A new approach towards block cipher design. In R. Anderson, editor, *Fast Software Encryption*, pages 18–33. Springer-Verlag, 1994.
10. J. Daemen, R. Govaerts, and J. Vandewalle. Weak keys of idea. In *Advances in Cryptology - CRYPTO'93*, pages 224–231. Springer-Verlag, 1994.
11. D.W. Davies. Some regular properties of the des. In *Advances in Cryptology - CRYPTO'82*, pages 89–96. Plenum Press, 1983.
12. L. Goubin and J. Patarin. Des and differential power analysis. In *CHES'99*, volume 1717, pages 158–172. Springer-Verlag, 1999.
13. J. Kelsey, B. Schneier, and D. Wagner. Key-schedule cryptanalysis of idea, g-des, gost, safer and triple-des. In *Advances in Cryptology - CRYPTO '96*, page 237. Springer-Verlag, 1996.
14. P. Kocher, J. Jaffe, and B. Jun. Introduction to differential power analysis and related attacks. *The article can be found at* `http://www.cryptography.com/dpa/technical/index.html`, 1998.
15. M. Matsui. Linear cryptanalysis method for des cipher. In *Advances in Cryptology - EUROCRYPT'93*, page 386. Springer-Verlag, 1993.

A First Order vs. Second Order DPA

Using a simple statistical model we compare the distinguishing power of second order DPA and first order DPA, i.e., we compute the required number of samples for both methods under identical noise levels.

Assume an emitter has chosen one of the two random process (with x a continuous variable, e.g., the power consumption at a given stage):

– Process f, probability density $f(x)$;
– Process g, probability density $g(x)$.

The observer receives a sequence of x_i and wants to determine whether it comes from f or from g. From the observer's point of view, the probability that values in $[x_i, x_i + dx_i]$ come from f is:

$$\begin{aligned} P(f \mid x_1 x_2 \ldots x_z)\; dx_1 \ldots dx_z &= \frac{P(x_1 x_2 \ldots x_z \mid f) P(f)}{P(x_1 x_2 \ldots x_z)}\; dx_1 \ldots dx_z \\ &= P(f) \prod_i \frac{P(x_i \mid f)}{P(x_i)}\; dx_i, \end{aligned}$$

and similarly for g,

$$P(g \mid x_1 x_2 \ldots x_z)\; dx_1 \ldots dx_z = P(g) \prod_i \frac{P(x_i \mid g)}{P(x_i)}\; dx_i.$$

For simplicity, we now assume that $P(f) = P(g)$, and thus one gets

$$\frac{P(f \mid x_1 x_2 \ldots x_z)}{P(g \mid x_1 x_2 \ldots x_z)} = \prod_i \frac{P(x_i \mid f)}{P(x_i \mid g)} = \prod_i \frac{f(x_i)}{g(x_i)}$$

In order to detect f over g, we must reach the situation where

$$P(f \mid x_1 x_2 \ldots x_z) \geq \lambda P(g \mid x_1 x_2 \ldots x_z).$$

For simplicity, let $\lambda = e$. Taking the logarithm on both sides, it reads:

$$\sum_i (\log(f(x_i)) - \log(g(x_i)) \geq 1 \tag{11}$$

The main question we address is how many samples (parameter z) are required to reach this condition. Assuming that the emitter chose the random process f, i.e., distribution $f(x)$ applies, each new x_i will on average contribute to the sum in (11) as much as:

$$D(f \| g) = \int f(x)(\log(f(x)) - \log(g(x))dx,$$

where $D(f\|g)$ happens to be the relative entropy of f and g (see [6]).

Therefore, the average number of samples required to clearly distinguish f from g is $z \sim 1/D(f\|g)$.

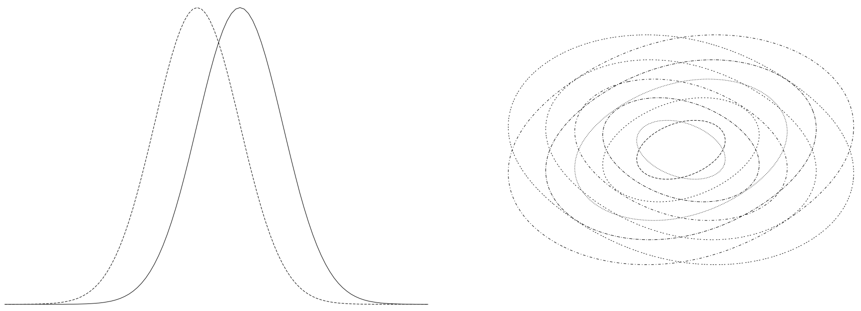


Fig. 1. First order vs. second order DPA distributions. The 2D distributions clearly have more overlap than 1D distributions.

The above result does not depend on a particular form of $f(x)$ or $g(x)$. We will now illustrate this with normal distributions:

- First order DPA: One has to distinguish between two noisy sets with slightly different averages. We thus create two classes, one for logical zero $f \sim N(0,\sigma)$ and one for logical one $g \sim N(1,\sigma)$. See Figure 1 (left).
- Second order DPA: The same individual distributions are used. However, one is interested in the exor of two independent bits. Therefore, we create two 2D distributions. See Figure 1 (right).
  - Since $0 = 0 \oplus 0 = 1 \oplus 1$, f consists of a balanced mix of $N(0,\sigma) \times N(0,\sigma)$ and $N(1,\sigma) \times N(1,\sigma)$.
  - Since $1 = 0 \oplus 1 = 1 \oplus 0$, g consists of a balanced mix of $N(0,\sigma) \times N(1,\sigma)$ and $N(1,\sigma) \times N(0,\sigma)$.

Notice that symmetry implies $D(f\|g) = D(g\|f)$ in this case. We numerically evaluate $D(f\|g)$ with varying σ for both first order and second order DPA. The results are listed in the table below. It appears that the number of samples necessary for second order DPA is much higher than that of first order DPA with the approximate relationship $z_{2^{nd}\mathrm{ODPA}} \approx 2z^2_{1^{st}\mathrm{ODPA}}$.

σ	$z_{1^{st}\mathrm{ODPA}}$	$z_{2^{nd}\mathrm{ODPA}}$
4	8	143
5.7	16	543
8	32	2111
11.3	64	8319
16	128	33023
22.6	256	131586
32	512	525326

Securing the AES Finalists Against Power Analysis Attacks

Thomas S. Messerges

Motorola Labs, Motorola
1301 E. Algonquin Road, Room 2712, Schaumburg, IL 60196
tomas@ccrl.mot.com

Abstract. Techniques to protect software implementations of the AES candidate algorithms from power analysis attacks are investigated. New countermeasures that employ random masks are developed and the performance characteristics of these countermeasures are analyzed. Implementations in a 32-bit, ARM-based smartcard are considered.

1 Introduction

The field of candidates for the final round of the Advanced Encryption Standard (AES) selection process has been narrowed from fifteen down to five finalists: Mars [1], RC6 [2], Rijndael [3], Serpent [4], and Twofish [5]. The cryptographic strength of the remaining AES candidates is currently being evaluated and the winning algorithm will soon be selected. One would expect that a cryptosystem using the AES winning algorithm would be unbreakable. However, history has proven that otherwise secure cryptographic algorithms can often succumb to weaknesses in their implementations [6]. Attackers of the AES algorithm may try to exploit such weaknesses.

Attacks on implementations are of particular concern to issuers and users of smartcards. Smartcards are becoming a preferred way of securely managing applications in industries such as telecommunications [7], health care [8], transportation [9], pay-TV [10] and internet commerce [11]. Smartcards have also been suggested for use in security applications such as network access [12] and physical access to locations such as automobiles, homes, and businesses [13]. Smartcards, however are potentially vulnerable to implementation attacks. These attacks include power analysis attacks [14,15], timing attacks [16,17], fault insertion attacks [18,19], and electromagnetic emission attacks [20]. All of these attacks exploit the fact that a hardware device can sometimes leak information when running a cryptographic algorithm. Kelsey et al. [21] use the term “side-channel” to describe this unintended leakage of information.

In a power analysis attack the side-channel information is the device’s power consumption. The power consumption of a vulnerable device, such as a smartcard, can leak information about the secrets contained inside the device. Kocher et al. first described power analysis attacks against the DES algorithm in a 1998 technical report [22] and later followed up with a paper presented at CRYPTO ‘99 [14]. Researchers have also begun to study the vulnerabilities of public-key cryptosystems to these attacks [23, 24]. Power analysis attacks against the AES algorithms have also been studied [25-27]. The purpose of this paper is to introduce and describe new implementations of the AES candidate algorithms that are secure against power analysis attacks. I present masking techniques that are used to randomize the fundamental operations used by the AES algorithms. I also provide results showing the performance and memory requirements for these implementations in a 32-bit, ARM-based smartcard.

1.1 Research Motivation

Tamper-resistant devices, such as smartcards, can be used to store and apply secret keys in symmetric-key cryptosystems. In a simple transaction, the smartcard might prove its authenticity through a basic challenge-and-response protocol. In such a protocol, an external device, called a reader, will challenge the smartcard to encrypt a random nonce. The smartcard will then use its secret key to encrypt this nonce and produce a response. Since the reader and smartcard share the same secret key, the reader can examine the response and verify whether the smartcard is authentic.

Honest readers possess a copy of the smartcard's secret key, thus these readers will be able to verify the result of a challenge. However, dishonest readers will not know the value of the smartcard's secret key. Nevertheless, when the power consumption leaks information from a smartcard, a dishonest reader might be able to ascertain the value of the secret key. Successful attacks against a smartcard's secret key might enable fraudulent behavior such as the counterfeiting of smartcards. Thus, smartcard issuers and users will want to ensure that the power consumption information cannot reveal the value of a secret key.

Future smartcard cryptosystems will likely use the AES algorithm. Thus, it is vital to begin understanding the issues involved in protecting smartcard AES implementations from attack.

1.2 Previous Work

The topic of power analysis attacks against the AES algorithms was discussed in March 1999 during the second AES Candidate Conference. At this conference, Biham and Shamir [25] submitted a paper that describes ways to attack many of the AES algorithms' key scheduling routines. In their paper, Biham and Shamir use the fact that a power consumption signal may leak the Hamming weight of the data being processed. They show how knowledge of the Hamming weights can enable attacks. In another paper, Chari et al. [26] look at attacks against the encrypt and decrypt routines of the AES candidate algorithms. Chari et al. assess each of the AES algorithm's vulnerabilities and, as an example, give results from an actual power analysis attack against a naive implementation of the Twofish algorithm. Daemen and Rijmen [27] also look at power analysis attacks against the AES algorithms. In their paper, Daemen and Rijmen examine the fundamental operations used by the AES algorithms and comment on possible vulnerabilities. Daemen and Rijmen also propose some possible countermeasures against these attacks.

All three of these papers recommend that comparisons of smartcard AES implementations consider the performance of versions that are secured against power analysis attacks rather than merely naive implementations.

1.3 Paper Overview

A smartcard microprocessor has a minimal amount of computing power and memory. Unfortunately, as pointed out by Chari et al. [26], software countermeasures against power analysis attacks can result in significant memory and execution time overhead. The amount of overhead seems to depend on the type and arrangement of the fundamental operations used by an algorithm. In this paper I examine the fundamental operations used by each of the AES finalist algorithms. I then develop techniques that use random masks to make software implementations of these operations resistant to power analysis attacks. Finally, I use these new countermeasures to implement masked versions of each of the remaining AES algorithms. The performance and implementation characteristics of these countermeasures in a 32-bit, ARM-based smartcard are analyzed.

The organization of this paper is as follows; first in Section 2, the fundamental operations used by each of the AES finalist algorithms are described. Next in Section 3, the basic principles of power analysis attacks are reviewed and previously suggested countermeasures are discussed. Then in Sections 4 and 5, my specific countermeasures for the AES finalist algorithms are described and implementation details are provided. Finally, the results for secure implementations of each of the algorithms are reported in Section 6.

2 Fundamental Operations in the AES Algorithms

The fundamental operations used by the AES algorithms were previously summarized by Daemen and Rijmen [27]. I review these fundamental operations and make the cautious assumption that each of these operations is potentially vulnerable to some form of power analysis attack. I then convert these vulnerable operations into secured operations using a strategy that employs random masks. Finally, I use these secured operations as the building blocks for the AES algorithms. Daemen and Rijmen [27] also proposed countermeasures, however their countermeasures are different from the masking strategies described in this paper. Daemen and Rijmen suggest software countermeasures such as data balancing and instruction sequence scrambling, whereas my countermeasures involve masking all intermediate data with random masks.

The AES candidate algorithms share many of the same fundamental operations. These operations include table lookups, bitwise AND, OR and XOR functions, shift and rotate operations, multiplication and addition modulo 2^{32} operations, permutations, polynomial multiplications over GF(2^8), and other various types of linear transformations. In this paper, I consider only the fundamental operations that are needed for implementations in a smartcard. Smartcards have a severely limited amount of memory, especially RAM, so some operations that could make the algorithms run more efficiently in a memory-rich environment are not considered in this paper.

A detailed description of the fundamental operations used in smartcard implementations of the AES algorithms is now given. A summary of these fundamental operations is also provided in Table 1.

2.1 Table Lookup Operations

All of the AES finalist algorithms, except the RC6 algorithm and a bitslice implementation of the Serpent algorithm, require table lookup operations. A table lookup operation operates on n input bits and produces m output bits. The n input bits specify an address into a table and the data at this address is the m-bit output. The size of the table grows exponentially with n, thus for smartcard implementations n must be kept fairly small. A table lookup operation T that has x as input and y as output is symbolically denoted as $y = T[x]$. The countermeasures that I will propose in Section 4 require the table be randomly masked prior to running the algorithm. This means that the table will need to be copied into RAM. Thus, table size is critical when considering secure implementations in smartcards with a minimal amount of RAM.

Mars. The Mars algorithm requires the largest table size out of all the AES algorithms. For Mars, $n = 9$ and $m = 32$ resulting in a table size of 2,048 bytes. During the forward and backwards mixing stages of Mars, this large table is viewed as two smaller tables each with $n = 8$ and $m = 32$.

Rijndael. The Rijndael algorithm actually does not need a table lookup operation because the table can be described arithmetically. However, this approach would result in a very inefficient implementation. Thus, for efficiency, Rijndael uses a table lookup operation where $n = 8$ and $m = 8$. The resulting table size is 256 bytes. More tables could be used for more efficient implementations, but these implementations are less suitable for low-memory smartcards.

Serpent. The Serpent algorithm can be implemented in either standard mode or in a more efficient bitslice mode. Only the standard mode requires table lookups. In this case, there are eight tables where $n = m = 4$. Thus, for standard mode Serpent, there is a need for 64 bytes of table memory.

Twofish. The Twofish algorithm can have its tables represented in a variety of forms. The tables are all originally derived from eight small permutation tables that have a combined memory requirement of 64 bytes. Implementations using these small tables would be inefficient, so more optimized implementations represent these small tables using two larger tables requiring a total of 512 bytes. More efficient implementations have been described that use key dependent tables [5], but these implementations are not suitable for low-memory smartcards.

2.2 Bitwise Boolean Functions

Bitwise Boolean functions include the AND, OR, and XOR functions. All of the AES algorithms use the XOR function. A bitslice implementation of the Serpent lookup tables and a routine to “fix” Mars subkeys are the only two places where the AND and OR functions are used. The logical operators $\oplus$, $\wedge$, and $\vee$ are used to denote the bitwise XOR, AND and OR functions, respectively. Smartcards can efficiently perform these operations, but if countermeasures are not taken, information regarding the operands and results may leak.

2.3 Shift and Rotate Operations

There are two types of shift and rotate operations, fixed and data dependent. All of the AES candidates use a fixed rotate or shift operation. Mars and RC6 also use data dependent rotate operations. Data dependent rotate operations can be very difficult to mask in smartcard implementations, especially if the smartcard microprocessor can only rotate by one bit at a time. Shift operations are denoted using the $\gg$ or the $\ll$ operator and rotate operations are denoted using the $\lll$ or the $\ggg$ operator.

2.4 Addition and Multiplication Modulo 2^{32}

Mars, RC6, and Twofish extensively use addition modulo 2^{32}. The Mars and RC6 algorithms also require a modular multiplication operation. In RC6, multiplication is used twice during each round. In Mars, multiplication is used once during each of the sixteen keyed transformation rounds.

2.5 Bitwise Permutations

A bitwise permutation is a rearrangement of the bits within a sequence of bits. The only AES algorithm that uses a bitwise permutation is the Serpent algorithm, and Serpent only uses the bitwise permutation if it is implemented in standard mode. A permutation operation P that has x as input and y as output is symbolically denoted as $y = \pi_P[x]$.

2.6 Polynomial Multiplications over GF(2^8)

The Rijndael and Twofish algorithm are the only AES finalists that use polynomial multiplications over GF(2^8). Polynomial multiplication can be implemented either directly or through the use of table lookup operations. When implemented directly, the multiplication decomposes into a series of conditional bitwise XOR operations and shifts. Conditional operations, however may allow for timing attacks. Therefore, smartcard implementations may instead use a combination of table lookups, XOR operations and shifts.

2.7 Linear Transformations

Serpent uses a linear transformation during each round and a recursive linear transformation during the key scheduling. Mars also uses a recursive linear transformation during the key schedule. Linear transformations can be implemented using shift and XOR operations, so techniques to protect these operations can also apply to linear transformations. A linear transform is symbolically denoted as $y = \mathrm{LT}[x]$, where x is the input to the transform and y is the output.

3 Review of Power Analysis Attacks and Countermeasures

Before looking at ways to securely implement the AES fundamental operations, it is useful to review the basic concepts of power analysis attacks. Kocher et al. [14] have described two types of attacks, a Simple Power Analysis (SPA) attack and a Differential Power Analysis (DPA) attack. An SPA attack is described as an attack where the adversary can directly use a single power consumption signal to break a cryptosystem. Attacks where an adversary can learn the Hamming weight of data that was processed or can learn how branch instructions were executed are examples of SPA attacks. The information in the power signal is usually quite small; thus steps such as executing random dummy code or avoiding memory accesses by processing data in registers can often help to protect against SPA attacks.

DPA attacks, on the other hand, can be much harder to protect against. A DPA attack uses statistical analysis to extract information from a power signal. Information that might be imperceptible by using SPA can often be extracted using DPA. In its minimal form, DPA reduces to the analysis of the probability distributions of points in the power consumption signal. For example, in the original DPA attack described by Kocher et al., the means of the probability distributions are analyzed.

Let $f(p)$ be the probability distribution function of a point in the power consumption signal that is vulnerable to attack. The underlying mechanism that enables a DPA attack is the fact that $f(p)$ can be dependent on the data input to the algorithm, the data output from the algorithm, and the secret key used by the algorithm. Most operations performed by an algorithm have this property, thus most operations are potentially vulnerable to a DPA attack.

Daemen and Rijmen [27] suggested software countermeasures against DPA attacks. These countermeasures include the insertion of dummy code, power consumption randomization, and the balancing of data. These methods will degrade the strength of a DPA attack, but may not be enough to prevent an attack. Chari et al. [28] suggest that ad hoc countermeasures will not suffice since attackers can theoretically use signal processing techniques to remove dummy code and can analyze more data to overcome the effects of randomization and data balancing. They suggest a better approach is to split all intermediate data results using a secret sharing scheme, thereby forcing attackers to analyze joint distribution functions on multiple points in the power signal. Goubin et al. [29] proposed a similar strategy, called the duplication method, to protect the DES algorithm from DPA

	Mars	**RC6**	**Rijndael**	**Serpent**	**Twofish**
Table-Lookup **(Table Size)**	two 8 to 32, or one 9 to 32 (2,048 bytes)	none (0 bytes)	one 8 to 8 (256 bytes)	none, or eight 4 to 4 (64 bytes)	eight 4 to 4, or two 8 to 8 (64 or 512 bytes)
Bitwise Boolean	XOR	XOR	XOR	XOR, AND, OR	XOR
Shift or Rotate Operation	Variable	Variable	Fixed		Fixed
Multiplication mod 2^{32}	X	X			
Addition mod 2^{32}	X	X			X
Multiplication GF(2^8)			X		X
Bitwise Permutation				standard mode	
Linear Transformation	X			X	

Table 1. Summary of the Fundamental Operations in the AES Finalist Algorithms
The fundamental operations used by the AES finalist algorithms are given in the above table. The memory requirements are also provided for the table lookup operations. These memory requirements are given assuming a typical smartcard implementation.

attacks. The countermeasures I propose for the AES algorithms mask all intermediate data results and are similar to those suggested by Chari et al. and Goubin et al.

Chari et al. [28] suggested that not all intermediate data in all rounds of an algorithm need to be masked. For example, they suggest that only the first and last four rounds of DES need to use their scheme. On the other hand, Fahn et al. [30] developed an Inferential Power Analysis (IPA) method that can even attack the middle rounds of an algorithm. My implementations take a conservative approach and mask all data for all rounds.

4 Secure Implementations of the AES Fundamental Operations

My implementations resist SPA attacks by avoiding branch instructions. Other steps to prevent SPA attacks can be taken as deemed necessary, but the main focus of my attention was on resisting DPA attacks. The DPA countermeasures that I implement use random masks to obscure the calculations made by the fundamental operations. The random masks force the power consumption signals to be uncorrelated with the secret key and the input and output data; thus DPA attacks will require analysis of joint probability distributions.

In this paper I work exclusively with 32-bit words and use two types of masking operations. One type, that I refer to as *Boolean* masking, uses the bitwise XOR operation as the mask operator. The other type, that I refer to as *arithmetic* masking, uses addition and subtraction modulo 2^{32} as the mask operator. As an example of each type of masking strategy consider the masking of a word x with a random mask r_x. The results of masking x using each strategy give the following masked values x' :

$$\text{Boolean mask: } x' = x \oplus r_x \quad \text{or} \quad \text{arithmetic mask: } x' = (x - r_x) \bmod 2^{32}$$

My overall strategy is to randomly mask the input data and key data prior to executing the algorithm. The algorithm is then executed using the masked data so all intermediate results of the algorithm are also masked. Since new masks are randomly chosen for each new run of the algorithm, simple statistical analysis of the algorithm's power consumption is inadequate for a successful attack. The only way attackers will be able to mount a statistical attack will be to look at joint probability distributions of multiple points in the power signal. Such an attack is referred to as a higher-order DPA [14] attack and is much more formidable to execute than a normal DPA attack.

The above masking strategy is possible if all of the fundamental operations of the AES algorithms can work with masked input data and produce masked output data. All operations, except addition and multiplication, can readily work with Boolean masked data. For addition and multiplication, arithmetic masking will be used. Many of the AES algorithms combine Boolean and arithmetic functions, thus a way to convert back and forth between Boolean masking and arithmetic masking is needed.

The conversion from one type of masking to another needs to be done in such a way to avoid vulnerabilities to DPA attacks. The algorithm shown in Fig. 1 gives one possible approach. In this approach, the unmasked data is x, the masked data is x', and the mask is r_x. The algorithm works by unmasking x using the XOR operation and then by arithmetically remasking x using modular subtraction. Of course, an unmasked x may be vulnerable to power analysis attack. Thus, a random value C is used to randomly select whether x or $\bar{x}$ is unmasked. A DPA attack using statistical analysis of the means will not work against this algorithm because the attacker does not know whether D or $\overline{D}$ is being processed at the circled statement in the algorithm. Attackers that can determine the value of C will be able to run a DPA attack, but finding out the value of C is an SPA attack and SPA attacks are protected against using other means. A similar algorithm to that given in Fig. 1 can be used to convert from arithmetic masking back to Boolean masking.

The following sections now describe how each of the fundamental operations can work with masked data.

4.1 Table Lookup Operations

Recall that a table lookup operation takes an input x and produces an output y such that $y = T[x]$. In order to mask a table lookup operation, the table itself needs to be masked. The easiest way to mask a table is to use an input mask r_{in} and an output mask r_{out}. A Boolean masked table T' can then be defined in terms of T, r_{in}, and r_{out} where

$$T'[x] = T[x \oplus r_{\text{in}}] \oplus r_{\text{out}}$$

The masked table T' takes inputs that are masked with r_{in} and produces outputs that are masked with r_{out}. Thus, the table lookup operation has been converted to an operation that takes masked data for inputs and produces masked data as outputs.

For practical implementations in a smartcard, the random values for r_{in} and r_{out} can be chosen at the beginning of the algorithm. These values can then be used to construct the masked table that will be stored in RAM. Now, anytime a table lookup operation needs to be performed, the input data can be masked with r_{in} and the masked table can be used. The output of the masked table will be masked with r_{out}, so it can either be unmasked to reveal the true output or reused in another secure operation.

Fortunately, most of the AES algorithms use tables that are small enough to fit into the RAM available in a smartcard. Mars is the only algorithm where this solution is likely to pose a problem.

```
BooleanToArithmetic( $x'$ , $r_x$ ){
    randomly select: C = 0 or C = -1
    B = C ⊕ $r_x$;           /* B = $r_x$ or B = $\bar{r}_x$                    */
    $\boxed{A = B \oplus x';}$           /* A = $x$ or A = $\bar{x}$                      */
    A = A - B;            /* A = $x - r_x$ or A = $\bar{x} - \bar{r}_x$              */
    A = A + C;            /* A = $x - r_x$ or A = $\bar{x} - \bar{r}_x - 1$          */
    A = A ⊕ C;            /* A = $x - r_x$                                 */
    return(A);}
```

Fig. 1. Algorithm to convert from Boolean to Arithmetic Mask
This algorithm takes masked data x' and mask r_x as input, and returns a masked value `A` such that $(A + r_x)$ is equal to $(x' \oplus r_x)$. The circled statement is where x or $\bar{x}$ is unmasked, depending on the value of C.

4.2 Bitwise Boolean Functions

The bitwise XOR operation trivially works with Boolean masked data. To compute the XOR of two masked operands, simply compute the XOR of the masked operands and then the XOR of their corresponding masks. Thus, if the masked operands x' and y' are masked with r_x and r_y, respectively, then the masked output is $z' = x' \oplus y'$ and the new mask is $r_z = r_x \oplus r_y$.

The bitwise AND operator can also be masked, but the calculation of the mask is a little more complicated. Again, the operands x' and y' are assumed to be Boolean masked with r_x and r_y, respectively. The masked output of the AND operation is $z' = x' \wedge y'$ and the mask can be shown to be $r_z = (r_x \wedge y') \oplus (r_y \wedge x') \oplus (r_x \wedge r_y)$. A similar expression can be derived for the bitwise OR operator.

A straightforward implementation of the above expression for r_z will first calculate $r_x \wedge y'$ and then use the XOR operation to combine this with $r_y \wedge x'$ or $r_x \wedge r_y$. Unfortunately, the intermediate result of this operation will cause some data of x or y to become unmasked. A simple fix is to use an intermediate random mask during these calculations.

4.3 Shift and Rotate Operations

Fixed rotate or shift operations can easily be performed on Boolean masked data. The masks simply rotate or shift along with the data. Thus, if the masked operand is x' and the mask is r_x, then the output of a right rotate by n is $x' \ggg n$ and the new mask is $r_x \ggg n$.

For data dependent rotate operations, the rotation amount also needs to be masked. This rotation amount, however, needs to be masked with an arithmetic mask rather than a Boolean mask. Thus, the data to be rotated is still represented as masked operand x' and Boolean mask r_x, but the rotation amount is now represented as masked operand n' and arithmetic mask r_n. A masked data dependent rotate operation can now be performed using two double rotations. The masked output of the right rotate operation is $(x' \ggg n') \ggg r_n$ and the corresponding mask is $(r_x \ggg n') \ggg r_n$.

4.4 Addition and Multiplication Modulo 2^{32}

The arithmetic operations of addition and multiplication are more compatible with arithmetic masking than with Boolean masking. The addition operation trivially works with arithmetic masked data. Given masked operands x' and y', which are masked with r_x and r_y, respectively, the masked output of the addition operation is simply $z' = (x' + y') \bmod 2^{32}$ and the new mask is $r_z = (r_x + r_y) \bmod 2^{32}$.

Multiplication of masked data is more involved. The masked result for multiplying masked operands x' and y' is $z' = (x'y') \bmod 2^{32}$ and the corresponding mask can be shown to be $r_z = (r_x y' + r_y x' + r_x r_y) \bmod 2^{32}$.

4.5 Bitwise Permutations

Bitwise permutations work very nicely with Boolean masked data. If the masked operand is x' and the mask is r_x, then the masked output is $z' = \pi_P[x']$ and the corresponding mask is $r_z = \pi_P[r_x]$.

4.6 Polynomial Multiplications over GF(2^8)

There are various ways that polynomial multiplications can work with Boolean masked data. If the multiplications are performed using table lookups, shift and XOR operations, then the corresponding methods to protect these operations can be used. Also, a data byte g that is Boolean masked with r_g using the XOR operation is equivalent to polynomial $g(x)$ being arithmetically masked with polynomial $r_g(x)$ using polynomial addition. Therefore, polynomial multiplication can be secured using an approach similar to that used for multiplication modulo 2^{32}.

4.7 Linear Transformations

Non-recursive linear transformations work nicely with masked data. Given a masked operand x' which is Boolean masked with r_x, the masked output is $z' = \mathrm{LT}[x']$ and the corresponding mask is $r_z = \mathrm{LT}[r_x]$. Recursive linear transformations can be represented as a series of shift and XOR operations, so the corresponding methods to protect shift and XOR operations can be used.

5 Implementation Details

The previously described masking techniques were used to implement secure versions of the five remaining AES candidates. Naive versions of the algorithms, without the use of masking, were also implemented as a baseline. It was very difficult to determine the best implementation methods from some of the algorithm specifications, so the details provided by the algorithm authors to NIST [31] proved to be useful references.

Each of the secured algorithms begin with an initialization step where random masks are generated and used to mask the input and key data. If needed, randomized tables are also constructed. The algorithms are then executed normally, except the masked data is processed with secure versions of the fundamental operations. The efficiency is reduced because the number of computations is increased and extra memory is required for the masks and the masked table data. Many operations need to be computed twice, once for the masked data and once for the masks. The table lookup operations also require extra overhead because the input data needs to be remasked using the mask that was originally used to construct the table. Some algorithms also require a significant amount of overhead to convert between Boolean and arithmetic masking.

5.1 Implementations for a Specific Processor

I chose to use a 32-bit, ARM-based processor as an evaluation platform for my AES implementations. The ARM processor, manufactured by ARM Ltd., is a RISC machine with fourteen general use registers. A smartcard containing an ARM processor typically has 4K bytes of RAM and 48K bytes of ROM. The ARM processor also has a barrel shifter and a 32-bit multiply instruction, both of which proved useful for my AES implementations. I wrote the code for the ARM processor completely in C and compiled the code using the C compiler that comes with the "ARM Software Development Kit" available from ARM Ltd. The code was compiled using the "-Otime" compiler flag, so the resulting machine code is optimized for time rather than size. More optimal code would likely be possible if some of the routines were written in assembly, but the intent of my experiment was to determine the relative costs of implementing secure code rather than producing the most efficient code possible.

I chose to implement my masking technique in a newer 32-bit smartcard processor, but could also have chosen an 8-bit processor. My software implementations required more than 256 bytes of RAM so low-memory 8-bit processors with only 256 bytes of RAM would not be suitable. However, newer 8-bit processors, such as the ST19 microprocessor manufactured by ST Microelectronics, would be sufficient for my implementations. An ST19 smartcard processor typically has 1K bytes of RAM and 32K bytes of ROM.

For information on 8-bit versus 32-bit implementations one could look at the work by Hachez et al. [32]. They examine implementations in both types of processors, so their results are useful for comparing 32-bit to 8-bit implementations. Comparisons of the Hachez et al. implementations to my implementations, however, should consider that the implementations by Hachez et al. were optimized to maximize performance, rather than to prevent power analysis attacks.

5.2 Algorithm Specific Issues

Masking the operations of the AES algorithms can be a very costly undertaking. The simplest approach to masking assigns each variable in the algorithm its own unique mask. Both the masks and the masked data need to be processed, thus both the amount of processing and the memory requirements double. In addition, there is also the cost associated with initializing the masks and storing and initializing masked lookup tables. Fortunately, there are a few techniques that can be used to reduce these costs.

One technique to save memory is to reuse the masks. In my implementations, two variables that are never directly combined are allowed to share the same mask, thus conserving valuable memory. Masks are also sometimes reused between rounds. For example, all the subkeys can often be masked with the same mask.

One technique to save processing is to start each round of the algorithm with a fixed set of masks. As the data for a round is manipulated, the masks will also change. In some cases, the changes to the masks are independent of the data being masked. Thus, in these cases the changes to the masks are predictable. A preprocessing step can calculate the intermediate mask values based on the initial masks and these values can be reused for every round. This technique reduces the need to continuously recalculate new masks during each round.

These techniques and other previously described masking techniques were used to implement the AES candidates. The main goal of these implementations was to keep the RAM memory requirements relatively low (less than 1K bytes) and the processing speed as fast as possible. The performance and memory requirements depend on the type of operations used in an algorithm and also the order of these operations. Comments on each of the AES implementations are now provided.

Mars. Mars was the most difficult algorithm to mask. The lookup table for Mars is 2K bytes, thus is unreasonably large to be masked and stored in RAM. As an alternative, two versions of the table were store in ROM. One table contained the normal unmasked data and the other table contained the complement data in reverse order. A random bit was used to determine which table to use. Other issues with Mars are that the multiplications are time consuming to mask and converting between Boolean and arithmetic masking is often required.

RC6. The RC6 algorithm, due to its simple design, was very easy to mask, but again multiplications and the need for repeatedly changing from Boolean to arithmetic masking caused a good deal of overhead. In general, the calculation of a mask for a multiplication operation requires three multiplies and two add operations, thus causing much overhead. Also, masks used in multiplications are dependent on the data being masked, thus these mask values cannot be precalculated. This prevents the use of the speedup technique based on precalculating the mask values.

Rijndael. The structure of the Rijndael algorithm made masking very efficient. There was no extra overhead for arithmetic operations and the lookup table was small enough to be masked and stored in RAM. During a Rijndael round, all operations on the masks proved to be independent of the data. Thus, the technique of precalculating mask values was used extensively in this implementation.

Serpent. The Serpent algorithm was only implemented in bitslice mode. An implementation in standard mode seemed too inefficient and potentially more vulnerable to a power analysis attack because individual bits would need to be processed during the permutation operation. The main source of overhead in the Serpent implementation is due to the bitwise AND and OR operations. Calculating the masks for one AND operation requires three AND operations and four XOR operations. The OR operation also requires two additional complement operations. Also, my technique to precalculate the mask values could not be used with the AND and OR operations. Thus, even though Serpent does not use costly arithmetic operations, the overhead was still relatively high.

Twofish. The Twofish algorithm uses arithmetic operations, but does not use multiplies. Thus, the masks can be precalculated. Also, the order of operations allowed for many masks to be shared. Overall, these properties led to a more efficient implementation. The main source of overhead in Twofish is the 512 bytes of RAM that is needed to store the masked lookup table.

6 Performance Measurements

Implementations of the encrypt mode of the AES algorithms were tested. The cycle counts and memory requirements for masked and unmasked implementations on the 32-bit ARM processor are given in Table 2. The security cost for each algorithm is also given in Table 2. The security cost was calculated by dividing the masked implementation result by the unmasked result.

It is clear that when the countermeasures described in this paper were used, some of the algorithms fared better than others. As expected, algorithms that used multiplication operations, such as Mars and RC6, showed the worst degradation in performance. Serpent is also not ideally suited for masking countermeasures, but its performance is a little more acceptable. Rijndael and Twofish are the best suited algorithms for random

Cycle Count	Mars	RC6	Rijndael	Serpent	Twofish
Unmasked	9,425	5,964	7,086	15,687	19,274
Masked	72,327	46,282	13,867	49,495	36,694
Security Cost	7.67	7.76	1.96	3.16	1.90

RAM (bytes)	Mars	RC6	Rijndael	Serpent	Twofish
Unmasked	116	232	52	176	60
Masked	232	284	326	340	696
Security Cost	2.00	1.22	6.27	1.93	11.60

ROM (bytes)	Mars	RC6	Rijndael	Serpent	Twofish
Unmasked	2,984	464	1,756	2,676	1,544
Masked	7,404	1,376	2,393	9,572	2,656
Security Cost	2.48	2.97	1.36	3.58	1.72

Table 2. Implementation Results for an ARM-based Processor
The implementation results above show the cycle count and memory requirements for the masked and unmasked versions of the AES algorithms. The security cost is calculated by dividing the masked implementation requirement by the unmasked requirement.

masking. The overall results show that masking countermeasures can be implemented in smartcards. The performance is degraded, but in some applications, security against power analysis attacks is more important than efficiency.

7 Conclusions

This paper has introduced various strategies for randomly masking the operations used by the AES finalists. These strategies were used in implementations of the AES algorithms and the performance of these implementations was reported. The results provide a useful means for comparing the efficiency of secure smartcard AES implementations. Perhaps future researchers can continue searching for more efficient secure implementation techniques. The efficiency of masking arithmetic operations especially needs to be addressed and secure implementations in hardware also need to be studied. Another approach may be to mask only critical operations, such as the first and last few rounds of an algorithm. Hopefully, the results of this paper can provide some initial guidance towards the selection of the winning AES algorithm and also assist in the future development of more secure software cryptosystems.

Acknowledgments

I would like to thank my colleagues Ezzy Dabbish and Louis Finkelstein of Motorola Labs for their helpful editing. Ezzy also provided many useful discussions regarding the AES implementations. I would also like to thank my Ph.D. advisor Professor Robert H. Sloan of the University of Illinois at Chicago. Professor Sloan provided encouragement and advice for writing this paper and also suggested many edits that improved the quality of the paper.

References

1. Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr., Luke O'Connor, Mohammad Peyravian, David Safford and Nevenko Zunic, "MARS – a candidate cipher for AES," IBM Corporation, AES submission available at: http://www.nist.gov/aes.
2. Ronald L. Rivest, M.J.B. Robshaw, R. Sidney and Y.L. Yin, "The RC6 Block Cipher," AES submission available at: http://www.nist.gov/aes.
3. Joan Daemen and Vincent Rijmen, "The Rijndael Block Cipher," AES submission available at: http://www.nist.gov/aes.
4. Ross Anderson, Eli Biham and Lars Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," AES submission available at: http://www.nist.gov/aes.
5. Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall and Niels Ferguson, "Twofish: A 128-Bit Block Cipher," AES submission available at: http://www.nist.gov/aes.
6. Ross Anderson, "Why Cryptosystems Fail," in Proceedings of *1st ACM Conference on Computer and Communications Security*, ACM Press, November 1993, pp. 215-227.
7. R. Mitchell, "The Smart Money is on Smart Cards: Digital Cash for Use in Pay Phones," *Business Week*, no. 3437, August 14, 1995, p. 68.
8. D. Maloney, "Progress of Card Technologies in Health Care," *CardTech/SecurTech 1998 Conference Proceedings*, Vol. 2, April 1998, pp. 333-351.
9. D. Fleishman, "Transit Cooperative Research Program Study: Potential of Multipurpose Fare Media," *CardTech/SecurTech 1998 Conference Proceedings*, Vol. 2, April 1998, pp. 755-769.
10. David M. Goldschlag and David W. Kravitz, "Beyond Cryptographic Conditional Access," *Proceedings of USENIX Workshop on Smartcard Technology*, May 1999, pp. 87-91.
11. R. J. Merkert, Sr., "Using Smartcards to Control Internet Security," *CardTech/SecurTech 1999 Conference Proceedings*, May 1999, pp. 815-824.
12. N. Itoi and P. Honeyman, "Smartcard Integration with Kerberos V5," *Proceedings of USENIX Workshop on Smartcard Technology*, May 1999, pp. 51-61.
13. F. J. Valente, "Tracking Visitors in the Brazilian Coffee Palace Using Contactless Smartcards," *CardTech/SecurTech 1998 Conference Proceedings*, Vol. 2, April 1998, pp. 307-313.
14. Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential Power Analysis," *Proceedings of Advances in Cryptology–CRYPTO '99*, Springer-Verlag, 1999, pp. 388-397.
15. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan, "Investigations of Power Analysis Attacks on Smartcards," *Proceedings of USENIX Workshop on Smartcard Technology*, May 1999, pp. 151-161.
16. Paul Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Proceedings of Advances in Cryptology–CRYPTO '96*, Springer-Verlag, 1996, pp. 104-113.
17. J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestré, J-J. Quisquater and J. L. Willems, "A Practical Implementation of the Timing Attack," in *Proceedings of CARDIS 1998*, Sept. 1998.
18. D. Boneh and R. A. Demillo and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in *Proceedings of Advances in Cryptology–Eurocrypt '97*, Springer-Verlag, 1997, pp. 37-51.
19. Eli Biham and Adi Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," in *Proceedings of Advances in Cryptology–CRYPTO '97*, Springer-Verlag, 1997, pp. 513-525.

20. W. van Eck, “Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk,” *Computers and Security*, v. 4, 1985, pp. 269-286.
21. J. Kelsey, B. Schneier, D. Wagner, and C. Hall, “Side Channel Cryptanalysis of Product Ciphers,” in *Proceedings of ESORICS ‘98*, Springer-Verlag, September 1998, pp. 97-110.
22. Paul Kocher, Joshua Jaffe, and Benjamin Jun, “Introduction to Differential Power Analysis and Related Attacks,” http://www.cryptography.com/dpa/technical, 1998.
23. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan, “Power Analysis Attacks of Modular Exponentiation in Smartcards,” *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems*, Springer-Verlag, August 1999, pp. 144-157.
24. Jean-Sébastien Coron, “Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems,” *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems*, Springer-Verlag, August 1999, pp. 292-302.
25. Eli Biham, Adi Shamir, “Power Analysis of the Key Scheduling of the AES Candidates,” *Second Advanced Encryption Standard (AES) Candidate Conference*, http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm, March 1999.
26. S. Chari, C. Jutla, J.R. Rao, P. Rohatgi, “A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards,” *Second Advanced Encryption Standard (AES) Candidate Conference*, http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm, March 1999.
27. Joan Daemen and Vincent Rijmen, “Resistance Against Implementation Attacks: A Comparative Study of the AES Proposals,” *Second Advanced Encryption Standard (AES) Candidate Conference*, http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm, March 1999.
28. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao and Pankaj J. Rohatgi, “Towards Sound Approaches to Counteract Power-Analysis Attacks,” *Proceedings of Advances in Cryptology–CRYPTO ‘99*, Springer-Verlag, 1999, pp. 398-412.
29. Louis Goubin and Jacques Patarin, “DES and Differential Power Analysis – The Duplication Method,” *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems*, Springer-Verlag, August 1999, pp. 158-172.
30. Paul N. Fahn and Peter K. Pearson, “IPA: A New Class of Power Attacks,” *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems*, Springer-Verlag, August 1999, pp. 173-186.
31. NIST, “CD-3: AES Finalists,” http://csrc.nist.gov/encryption/aes/round2/aescdrom.htm, October 1999.
32. G. Hachez, F. Koeune, J-J. Quisquater, “cAESar Results: Implementation of Four AES Candidates on Two Smart Cards,” *Second Advanced Encryption Standard (AES) Candidate Conference*, http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm, March 1999.

Ciphertext only Reconstruction of Stream Ciphers Based on Combination Generators

Anne Canteaut[1] and Eric Filiol[1,2]

[1] INRIA, projet CODES, Domaine de Voluceau
78153 Le Chesnay Cedex, FRANCE
{Anne.Canteaut,Eric.Filiol}@inria.fr
[2] Ecoles militaires de Saint-Cyr Coëtquidan,
DGER/CRECSC/DSI, 56381 Guer Cedex, FRANCE
efiliol@mailhost.esm-stcyr.terre.defense.gouv.fr

Abstract. This paper presents an operational reconstruction technique of most stream ciphers. We primarily expose it for key-stream generators which consist of several linear feedback shift registers combined by a nonlinear Boolean function. It is shown how to completely recover the different feedback polynomials and the combining function, when the algorithm is totally unknown. This attack only requires the knowledge of some ciphertexts, which may be generated from different secret keys. Estimates of necessary ciphertext length and experimental results are detailed.

Keywords: stream cipher, Boolean function, correlation, linear feedback shift register, ciphertext only reconstruction

1 Introduction

Stream ciphers are an important class of cipher systems. They are widely used by the world's militaries and governmental offices. They also are very often used in industrial encryption products. The success of stream ciphers comes from the fact that they are very easy to build: they need only few logic gates in VLSI circuitry. They are therefore particularly appropriate to embedded systems (satellites for example) or to the systems for which maintenance is either impossible or very difficult. Moreover, their use is particularly well-suited when errors may occur during the transmission because they avoid error propagation.

In a binary additive stream cipher, the ciphertext is obtained by adding bitwise the plaintext to a pseudo-random sequence called the running-key (or the key-stream). The running-key is produced by a pseudo-random generator whose initialization is the secret key shared by the users. Most practical designs of pseudo-random generators center around *linear feedback shift registers* (LFSRs) combined by a nonlinear Boolean function. Different variants can actually be found: clock-controlled systems, filter generators, multiplexed systems...[13]. We here focus on the most common class of combination generators depicted in Figure 1.

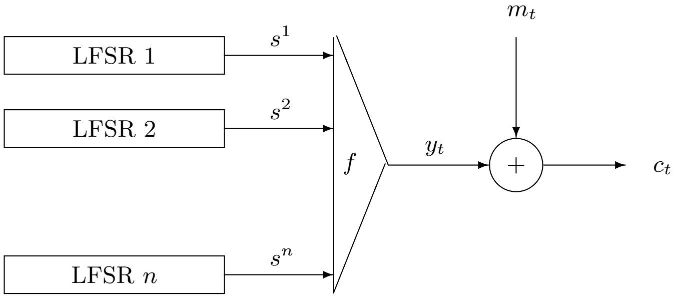


Fig. 1. Additive stream cipher using a combination generator

The other very important aspect is that the designs are often secret and contrary to block ciphers, generally no public evaluation is possible. Although such stream ciphers may be vulnerable to some attacks [11,7,6], cryptanalysis becomes much harder when the algorithm is unknown. During World War II, US cryptanalysts had to face this problem with the Japanese PURPLE machine [8]: they reconstructed it before cryptanalysing it. This paper presents a similar approach and a reconstruction technique of stream ciphers allowing, from ciphertexts only, complete recovering of the unknown algorithm. By algebraic and statistical results, all the cryptographic primitives constituting the system (the LFSR characteristics and the combining function) can be recovered. After this reconstruction step, the LFSR initializations can be found by classical correlation attacks [11,7,6,1].

The reconstruction has been conducted on the following basis and assumptions:

– The system is a combination generator. Most practical designs use combining functions with up to 5 or 7 variables (i.e., registers). In this paper we will only consider additive stream ciphers but generalization to other combining functions can be envisaged with suitable modifications.
– We use only ciphertexts, possibly generated from different secret keys. Each of them, however, must be of a realistic length.
– We know the plaintext encoding (or at least some of its statistical parameters) and the linguistic group of the plaintext language.
– We accept very long computing time since work is done only once (and for all) and as long as it remains far lower than the life of the algorithm itself.

This paper is organized as follows. Section 2 presents the theoretical tools we use in the reconstruction. In Section 3, we show how to recover the LFSRs and we give some simulation results. We precisely analyze the complexity of this attack and we estimate the number of required ciphertext bits. Section 4 focuses on the combining function recovering.

2 Theoretical Background

2.1 Linear Feedback Shift Register Sequences

A linear feedback shift register of length L is characterized by a univariate polynomial P over $\mathbf{F}_2$ of degree L, called the *feedback polynomial*, $P(x) = 1 + \sum_{i=1}^{L} p_i X^i$. It associates to any L-bit initialization $(s_t)_{1 \leq t \leq L}$ a sequence $(s_t)_{t>0}$ defined by the L-th order linear recurrence relation.

$$s_{t+L} = \sum_{i=1}^{L} p_i s_{t+L-i}, \quad t \geq 0 \ .$$

Most applications use a primitive feedback polynomial since this ensures that the periods of all sequences produced by the LFSR are maximal.

We now recall some well-known properties on LFSR sequences. In the following, $\mathcal{S}(P)$ denotes the set of all sequences produced by the LFSR with feedback polynomial P.

Proposition 1. [15,5,14] *Let P and Q be two non constant polynomials over $\mathbf{F}_2$. Then we have*

- $\{(u_t + v_t)_{t>0},\ u \in \mathcal{S}(P), v \in \mathcal{S}(Q)\} = \mathcal{S}(R)$ *where R is the least common multiple of P and Q.*
- $\{(u_t v_t)_{t>0},\ u \in \mathcal{S}(P), v \in \mathcal{S}(Q)\} = \mathcal{S}(R)$ *where* $\deg(R) \leq \deg(P)\deg(Q)$. *Equality holds if and only if at least one of the polynomials P and Q has only simple roots and all products $\alpha\beta$ are distinct for all α and β such that $P(\alpha) = 0$ and $Q(\beta) = 0$ in a common splitting field. This condition is notably satisfied if P and Q have coprime orders.*

Proposition 2. [9, Th. 8.53] *Let P and Q be two non constant polynomials over $\mathbf{F}_2$. Then $\mathcal{S}(P)$ is a subset of $\mathcal{S}(Q)$ if and only if P divides Q.*

This proposition implies that if a sequence s is generated by a LFSR with feedback polynomial P, then it satisfies the recurrence relations (or parity-check equations) corresponding to PQ for any $Q \in \mathbf{F}_2[X]$.

For a given feedback polynomial P of degree L, we focus on all recurrence relations corresponding to the multiples of P of weight d, where d is small. A similar approach is used in fast correlation attacks [11,1,7]. The following formula (see e.g. [1]) provides an approximation of the average number $m(d)$ of multiples Q of P which have weight d and degree at most D, $Q(X) = 1 + \sum_{j=1}^{d-1} X^{i_j}$:

$$m(d) \simeq \frac{D^{d-1}}{(d-1)!2^L} \ . \tag{1}$$

2.2 Boolean Functions for Stream Ciphers

A Boolean function with n variables is a function from the set of n-bit words, $\mathbf{F}_2^n$, into $\mathbf{F}_2$. Such a function can be expressed as a unique polynomial in $x_1, \ldots,$ x_n, called its *Algebraic Normal Form* (ANF):

$$f(x_1, \ldots, x_n) = \sum_{u \in \mathbf{F}_2^n} a_u x^u, \quad a_u \in \mathbf{F}_2$$

where $u = (u_1, \ldots, u_n)$ and $x^u = x_1^{u_1} x_2^{u_2} \ldots x_n^{u_n}$. The coefficients a_u of the ANF can be obtained from the Möbius transform of f [10]:

$$a_u = \bigoplus_{x \preceq u} f(x) \tag{2}$$

where $\oplus$ denotes the addition over $\mathbf{F}_2$ and $\alpha \preceq \beta$ describes the partial ordering on the Boolean lattice. This means that $\alpha \preceq \beta$ if and only if $\alpha_i \leq \beta_i$ for all $1 \leq i \leq n$.

The *Walsh-Hadamard transform* of a Boolean function f refers to the Fourier transform of the corresponding sign function, $x \mapsto (-1)^{f(x)}$:

$$\forall u \in \mathbf{F}_2^n, \quad \widehat{\chi}_f(u) = \sum_{x \in \mathbf{F}_2^n} (-1)^{f(x)} (-1)^{u \cdot x}$$

where $u \cdot x$ denotes the usual scalar product. The Walsh coefficient $\widehat{\chi}_f(u)$ then estimates the Hamming distance between f and the affine function $u \cdot x + \varepsilon$, $\varepsilon \in \mathbf{F}_2$, both seen as Reed-Muller codewords [10].

A Boolean function is obviously completely characterized by its Walsh spectrum. The coefficients of the algebraic normal form of f can then be computed from its Walsh coefficients as follows.

Proposition 3. *Let f be a Boolean function with n variables and let $(a_u)_{u \in \mathbf{F}_2^n}$ denote the coefficients of its algebraic normal form, i.e.,*

$$f(x_1, \ldots, x_n) = \sum_{u \in \mathbf{F}_2^n} a_u x^u \; .$$

Then we have, for all $u \in \mathbf{F}_2^n$, $a_u = 2^{wt(u)-1} \left(1 - \frac{1}{2^n} \sum_{v \preceq \bar{u}} \widehat{\chi}_f(v)\right) \bmod 2$ where $\bar{u}$ denotes the bitwise completion to 1 and $wt(u)$ is the Hamming weight of u, i.e., the number of its non-zero components.

Proof. From Equation (2) we have for any $u \in \mathbf{F}_2^n$

$$\begin{aligned} a_u &= \sum_{x \preceq u} f(x) \bmod 2 = \sum_{x \preceq u} \frac{1}{2} \left(1 - (-1)^{f(x)}\right) \bmod 2 \\ &= 2^{wt(u)-1} - \frac{1}{2} \sum_{x \preceq u} (-1)^{f(x)} \bmod 2 \end{aligned}$$

Since the normalized Fourier transform is involutive, we have

$$\forall x \in \mathbf{F}_2^n,\ \ (-1)^{f(x)} = 2^{-n} \sum_{v \in \mathbf{F}_2^n} \widehat{\chi}_f(v)(-1)^{v\cdot x}\ .$$

By combining these relations, we deduce that

$$\begin{aligned} a_u &= 2^{wt(u)-1} - 2^{-n-1} \sum_{x \preceq u} \sum_{v \in \mathbf{F}_2^n} \widehat{\chi}_f(v)(-1)^{v\cdot x} \bmod 2 \\ &= 2^{wt(u)-1} - 2^{-n-1} \sum_{v \in \mathbf{F}_2^n} \widehat{\chi}_f(v) \left(\sum_{x \preceq u} (-1)^{v\cdot x} \right) \bmod 2\ . \end{aligned}$$

The set $E_u = \{x \in \mathbf{F}_2^n,\ x \preceq u\}$ is a linear subspace of $\mathbf{F}_2^n$ of dimension $wt(u)$. Its orthogonal, $E_u^\perp$, satisfies $E_u^\perp = E_{\bar{u}}$. It follows that

$$\sum_{x \preceq u} (-1)^{v\cdot x} = \begin{cases} 2^{wt(u)} & \text{if } v \in E_{\bar{u}}, \\ 0 & \text{otherwise.} \end{cases}$$

We then obtain that, for all $u \in \mathbf{F}_2^n$,

$$a_u = 2^{wt(u)-1} - 2^{-n-1+wt(u)} \sum_{v \preceq \bar{u}} \widehat{\chi}_f(v) \bmod 2\ .$$

This proposition will be used in the attack for recovering the algebraic normal form of the combining function.

It is well-known that a combining function must fulfill some criteria to yield a cryptographically secure combination generator (see e.g. [3]). Most notably, combination generators are vulnerable to "divide-and-conquer" attacks, called *correlation attacks* [17]. These techniques fail when the combining function has a high correlation-immunity order [16].

Definition 1. *A Boolean function is* t-th order correlation-immune *if the probability distribution of its output is unaltered when any t input variables are fixed.*

This property equivalently asserts that the output of f is statistically independent of any linear combination of t input variables. Correlation-immunity can be characterized by the Walsh spectrum of the function [18]: f is t-th order correlation-immune if and only if

$$\forall u \in \mathbf{F}_2^n,\ 1 \leq wt(u) \leq t,\ \ \widehat{\chi}_f(u) = 0\ .$$

Since any t-th order correlation-immune function is k-th order correlation-immune for any $k \leq t$, we call correlation-immunity order of a function f the highest integer t such that f is t-th order correlation-immune. Note that the correlation-immunity order of a function with n variables can not exceed $(n-1)$. This comes from Parseval's relation:

$$\sum_{u \in \mathbf{F}_2^n} \left(\widehat{\chi}_f(u)\right)^2 = 2^{2n}\ .$$

This equality also points out the existence of a trade-off between the correlation-immunity order and the nonlinearity of a function.

3 Recovering the LFSRs

We now show how the key-stream generator depicted in Figure 1 can be reconstructed from the knowledge of some ciphertext bits.

In the rest of the paper we use the following notation. n denotes the number of constituent LFSRs. L_i and P_i denote the length and the feedback polynomial of the i-th LFSR and s^i refers to the generated sequence. The sequences y, m and c respectively correspond to the key-stream, to the plaintext and to the ciphertext. When dealing bitwise, we use t as index time.

The plaintext is assumed to be the output of a binary memoryless source with $P[m_t = 0] = p_0 \neq \frac{1}{2}$. All commonly used coding scheme (ASCII, Murray, CCITTx ...) satisfy this hypothesis. Moreover, the value of p_0 is supposed to be known. Practical values of p_0 are usually greater than 0.6.

The first step of the reconstruction consists in recovering the feedback polynomials of the constituent LFSRs.

3.1 Statistical Model

We first point out that the knowledge of a sequence s which is correlated with the ciphertext sequence provides some information on the feedback polynomials of the constituent LFSRs.

Proposition 4. *Let s be a binary sequence. If $P[c_t = s_t] \neq 1/2$ then there exists a Boolean function g with n variables such that $s = g(s^1, \ldots, s^n)$. Moreover, we have $P[c_t = s_t] = 1 - p_0 - p_g + 2p_0p_g$ where $p_g = P[f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)]$.*

Proof. We obviously have

$$\begin{aligned} P[c_t = s_t] &= P[y_t = s_t]P[m_t = 0] + P[y_t = s_t \oplus 1]P[m_t = 1] \\ &= 1 - p_0 - P[y_t = s_t] + 2p_0 P[y_t = s_t] \ . \end{aligned}$$

By hypothesis, $p_0 \neq 1/2$. Thus $P[c_t = s_t] \neq 1/2$ implies that $P[y_t = s_t] \neq 1/2$. Since $y = f(s^1, \ldots, s^n)$, the sequences y and s are statistically independent if s is statistically independent of $(s^1, \ldots, s^n)$. It follows that $P[y_t = s_t] = 1/2$ unless $s = g(s^1, \ldots, s^n)$ for some Boolean function g. In this case, we have

$$P[y_t = c_t] = P[f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n)] \ .$$

Note that some variables may not appear in the algebraic normal form of g.

If s is such that $P[c_t = s_t] \neq 1/2$ we deduce from the previous proposition and from Proposition 1 that the feedback polynomial of s is related to the feedback polynomials $P_1, \ldots, P_n$.

Corollary 1. *Let $\mathcal{S}(Q)$ denote the set of all sequences generated by $Q \in \mathbf{F}_2[X]$. If there exists $s \in \mathcal{S}(Q)$ such that $P[c_t = s_t] \neq 1/2$, then there exists a divisor Q' of Q and a Boolean function g such that Q' is derived from $P_1,\ldots,P_n$ and from g as described in Proposition 1.*

This result leads to the following algorithm for recovering some information on $P_1, \ldots ,P_n$. Let $\mathcal{Q}$ be a subset of $\mathbf{F}_2[X]$. For each $Q \in \mathcal{Q}$, we determine whether $\mathcal{S}(Q)$ contains a sequence which is correlated with the ciphertext. If such a sequence exists, Q provides some information on $P_1,\ldots,P_n$. We here choose for $\mathcal{Q}$ the set of all polynomials of $\mathbf{F}_2[X]$ of weight d and of degree at most D having the following form $Q(X) = 1+\sum_{j=1}^{d-1} X^{i_j}$. Recall that the degree of the feedback polynomial of the product of two sequences s^i and s^j is usually much higher than the degree of the feedback polynomial of $s^i + s^j$. If the upper-bound D on the degree of the examined polynomials is well-chosen, the polynomials Q detected by the algorithm then correspond to the case where the combining function g is linear. For $g(x) = u \cdot x$, any feedback polynomial of $s = g(s^1, \ldots, s^n)$ is a multiple of $\mathrm{lcm}_{i\in supp(u)} P_i$ where $supp(u) = \{i, u_i = 1\}$. Since all feedback polynomials are usually primitive, we have $\mathrm{lcm}_{i\in supp(u)} P_i = \prod_{i\in supp(u)} P_i$ in most practical situations. Moreover, we have

$$P[c_t = s_t] = \frac{1}{2} + \frac{(2p_0 - 1)}{2^{n+1}} \widehat{\chi}_f(u) \ . \tag{3}$$

Example 1. We consider the combination generator described by Geffe [4]. It consists of three LFSRs combined by the Boolean function $f(x_1, x_2, x_3) = x_1x_2+ x_2x_3 + x_1$. Assume that the feedback polynomials of the constituent LFSRs are randomly chosen primitive polynomials and that their lengths are respectively $L_1 = 15$, $L_2 = 17$ and $L_3 = 23$. Let c be the ciphertext sequence obtained by adding the output of Geffe generator to a plaintext with $p_0 \neq 0.5$. Let $\mathcal{Q}$ be the set of all polynomials of weight 4 and of degree at most 10000. For all $Q \in \mathcal{Q}$, we determine whether $\mathcal{S}(Q)$ contains a sequence which is correlated with c. We deduce from Formula (1) that, for a randomly chosen polynomial P of degree L, $\mathcal{Q}$ contains a multiple of P of weight 4 if $L \leq 37$. Our algorithm is then expected to detect multiples of P_1, P_2, P_3 and P_1P_2. Note that P_2 can not be detected by the algorithm since the Walsh coefficient $\widehat{\chi}_f(0, 1, 0)$ vanishes.

A simple method for determining whether $\mathcal{S}(Q)$ contains a sequence which is correlated with c consists in computing the parity-check equation corresponding to Q for the ciphertext bits. The efficiency of this procedure strongly depends on the weight of Q.

Theorem 1. *Let Q be a polynomial in $\mathbf{F}_2[X]$ of weight d having the following form $Q(X) = 1 + \sum_{j=1}^{d-1} X^{i_j}$ with $i_1 < i_2 < \ldots < i_{d-1}$. For a given ciphertext subsequence $(c_t)_{t<N}$ we consider the binary sequence $(z_t)_{i_{d-1}\leq t<N}$ defined by*

$$z_t = c_t \oplus \bigoplus_{j=1}^{d-1} c_{t-i_j} \ .$$

Then the random variable $Z = \sum_{t=i_{d-1}}^{N-1} (-1)^{z_t}$ has a Gaussian distribution with mean value $M = \pm(N - i_{d-1})(2\varepsilon)^d$ and with variance $\sigma^2 = (N - i_{d-1})(1 - (2\varepsilon)^{2d})$ where $\varepsilon = \max_{s \in \mathcal{S}(Q)} |P[c_t = s_t] - \frac{1}{2}|$.

Proof. Let $s \in \mathcal{S}(Q)$ be such that $|P[c_t = s_t] - \frac{1}{2}|$ is maximal. Let $p = P[c_t = s_t]$. For all t, we decompose c_t as $c_t = s_t \oplus e_t$ where $P[e_t = 1] = 1 - p$. Then we have

$$P[z_t = 1] = P[c_t \oplus \bigoplus_{j=1}^{d-1} c_{t-i_j} = 1] = P[e_t \oplus \bigoplus_{j=1}^{d-1} e_{t-i_j} = 1]$$

since s satisfies the recurrence relation because $s \in \mathcal{S}(Q)$. This implies that $z_t = 1$ if and only if the number of indexes $i \in \{t, t - i_1, \ldots, t - i_{d-1}\}$ such that $e_i = 1$ is odd. Therefore we have

$$\begin{aligned} P[z_t = 1] &= \sum_{\ell=0, \ell \text{ odd}}^{d} \binom{d}{\ell} (1-p)^\ell p^{d-\ell} \\ &= \frac{1}{2} \left[\sum_{\ell=0}^{d} \binom{d}{\ell} (1-p)^\ell p^{d-\ell} - \sum_{\ell=0}^{d} \binom{d}{\ell} (p-1)^\ell p^{d-\ell} \right] \\ &= \frac{1}{2} \left[1 - (2p-1)^d \right] \ . \end{aligned}$$

The random variable Z can now be expressed as $Z = (N - i_{d-1}) - 2 \sum_{t=i_{d-1}}^{N} z_t$. All random variables z_t are independent and identically distributed. Due to the central limit theorem [2], the random variable $\sum_{t=i_{d-1}}^{N} z_t$ for large values of $(N - i_{d-1})$ can be assumed to have a Gaussian distribution with mean value $(N - i_{d-1})P[z_t = 1]$ and variance $(N - i_{d-1})P[z_t = 1]P[z_t = 0]$. It follows that Z has a Gaussian distribution with mean value

$$M = (N - i_{d-1})(1 - 2P[z_t = 1]) = (N - i_{d-1})(2p - 1)^d$$

and with variance

$$\begin{aligned} \sigma^2 &= 4(N - i_{d-1})P[z_t = 1]P[z_t = 0] = (N - i_{d-1})(1 - (2p-1)^d)(1 + (2p-1)^d) \\ &= (N - i_{d-1})(1 - (2p-1)^{2d}) \ . \end{aligned}$$

If all sequences in $\mathcal{S}(Q)$ are statistically independent of c, Z has Gaussian distribution with mean value 0 and variance $(N - i_{d-1})$ since $\varepsilon = 0$ in this case.

We now want to distinguish between two hypotheses:

– $\mathcal{H}_0$: for all $s \in \mathcal{S}(Q)$, $P[c_t = s_t] = \frac{1}{2}$.
– $\mathcal{H}_1$: there exists $s \in \mathcal{S}(Q)$ such that $P[s_t = c_t] \neq \frac{1}{2}$.

We use a decision threshold T, $T > 0$, for discriminating hypotheses $\mathcal{H}_0$ and $\mathcal{H}_1$. If $|Z| < T$, $\mathcal{H}_0$ is kept; if $|Z| \geq T$, $\mathcal{H}_1$ is accepted. The minimum number of required ciphertext bits, N, depends on the number of wrong decisions that we allow. This number corresponds to the probability for a false alarm, $P_f = P[|Z| \geq T \mid \mathcal{H}_0]$. The decision threshold is determined by the probability for a non-detection, $P_n = P[|Z| < T \mid \mathcal{H}_1]$. Let Φ denotes the normal distribution function,

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{x^2}{2}\right) dx \ .$$

Then we have

$$P_f = P[|Z| \geq T \mid \mathcal{H}_0] = 2\Phi\left(\frac{-T}{\sqrt{N - i_{d-1}}}\right) \ .$$

Similarly the probability for a non-detection is given by

$$\begin{aligned} P_n = P[|Z| < T \mid \mathcal{H}_1] &= \frac{1}{\sqrt{2\pi}} \int_{\frac{-T-M}{\sigma}}^{\frac{T-M}{\sigma}} \exp\left(-\frac{x^2}{2}\right) dx \\ &= \Phi\left(\frac{T-M}{\sigma}\right) - \Phi\left(\frac{-T-M}{\sigma}\right) = \Phi\left(\frac{T-|M|}{\sigma}\right) - \Phi\left(\frac{-T-|M|}{\sigma}\right) \end{aligned}$$

since M is not necessarily positive. In most cases, $\Phi(\frac{-T-|M|}{\sigma})$ is much smaller than P_n and than $\Phi(\frac{T-|M|}{\sigma})$. Then this latter will approximate P_n. The predetermined value of P_n fixes the choice for the threshold:

$$T = |M| + \Phi^{-1}(P_n)\sigma = (N - i_{d-1})(2\varepsilon)^d + \Phi^{-1}(P_n)\sqrt{(N - i_{d-1})(1 - (2\varepsilon)^{2d})} \ .$$

Similarly, the predetermined probability for a false alarm gives the minimum value of $(N - i_{d-1})$:

$$N - i_{d-1} = \left(\frac{T}{\Phi^{-1}(1 - \frac{P_f}{2})}\right)^2 \ .$$

After different attempts to tune up the best values for P_f and P_n, we choose $P_f = 2^{-20}$ and $P_n = 10^{-3}$. In practical situations the known ciphertext sequence does not consist of a large number of consecutive bits. The attacker has access to some ciphertext blocks of reasonable lengths. These ciphertexts may be produced with different keys, i.e., with different LFSR initializations. Theorem 1 can nevertheless be adapted to this more realistic situation.

Corollary 2. *Let Q be a polynomial in $\mathbf{F}_2[X]$ of weight d having the following form $Q(X) = 1 + \sum_{j=1}^{d-1} X^{i_j}$ with $i_1 < i_2 < \ldots < i_{d-1}$. For n_c ciphertexts c^k, $1 \leq k \leq n_c$, of respective lengths $LC(k)$, we consider the binary sequence $(z_t^k)_{i_{d-1} \leq t < LC(k), 1 \leq k \leq n_c}$ defined by*

$$z_t^k = c_t^k \oplus \bigoplus_{j=1}^{d-1} c_{t-i_j}^k \ .$$

Then the random variable

$$Z = \sum_{k=1}^{n_c} \sum_{t=i_{d-1}}^{LC(k)-1} (-1)^{z_t^k}$$

has a Gaussian distribution with mean value

$$M = \pm(2\varepsilon)^d \sum_{k=1}^{n_c} (LC(k) - i_{d-1})$$

and with variance

$$\sigma^2 = (1 - (2\varepsilon)^{2d}) \sum_{k=1}^{n_c} (LC(k) - i_{d-1})$$

where $\varepsilon = \max_{s \in \mathcal{S}(Q)} |P[c_t = s_t] - \frac{1}{2}|$.

The following algorithm then examines all polynomials of degree at most D and of weight d, and it detects all polynomials Q in this set such that there exists $s \in \mathcal{S}(Q)$ with $|P[s_t = c_t] - 1/2| \geq \varepsilon_{\min}$.

Algorithm

```
For each $(d-1)$-tuples $(i_1, \ldots, i_{d-1})$ such that $0 < i_1 < \ldots < i_{d-1} < D$
    $N \leftarrow \sum_{k=1}^{n_c} (LC(k) - i_{d-1})$.
    $T \leftarrow N(2\varepsilon_{\min})^d - 3\sqrt{N(1 - (2\varepsilon_{\min})^{2d})}$.
    $Z \leftarrow 0$.
    For each ciphertext block $(c_t^k)_{0 \leq t < LC(k)}$ where $LC(k) > i_{d-1}$
        for each $t$ from $i_{d-1}$ to $LC(k) - 1$
            $z \leftarrow c_t^k \oplus \bigoplus_{j=1}^{d-1} c_{t-i_j}^k$.
            $Z \leftarrow Z + (-1)^z$.
    If $|Z| \geq T$, store $1 + \sum_{j=1}^{d-1} X^{i_j}$ and the value of $Z$.
```

Some gcd computations on the obtained polynomials provide the primitive factors which are detected several times. These primitive factors are expected to be the feedback polynomials of the constituent LFSRs.

3.2 Complexity Analysis

We now discuss the choice of the input parameters d, D and $\varepsilon_{\min}$.

Recall that we aim at recovering multiples of polynomials $\prod_{i \in T} P_i$, $T \subset \{1, \ldots, n\}$ such that $|P[c_t = \bigoplus_{i \in T} s_t^i] - 1/2| \geq \varepsilon_{\min}$. According to Formula (3), these subsets T are characterized by

$$\frac{|2p_0 - 1|}{2^{n+1}} |\widehat{\chi}_f(1_T)| \geq \varepsilon_{\min}$$

where the i-th component of 1_T equals 1 if and only if $i \in T$. It is well-known that all Walsh coefficients of a Boolean function f with n variables are divisible by 4, unless f has degree n. This case is here dismissed since such a function cannot be balanced. Choosing

$$\varepsilon_{\min} = \frac{|2p_0 - 1|}{2^{n-1}} \tag{4}$$

then ensures to detect all polynomials $\prod_{i \in T} P_i$ such that $\widehat{\chi}_f(1_T) \neq 0$. In most practical situations, the number of variables n does not exceed 7.

We now assume that our search can be restricted to all products $\prod_{i\in T} P_i$ of degree at most $L_{\max}$. This means that we suppose that all feedback polynomials $P_1,\ldots,P_n$ can be recovered from all products $\prod_{i\in T} P_i$ such that $\widehat{\chi}_f(1_T) \neq 0$ and $\sum_{i\in T} L_i \leq L_{\max}$. Note that $L_{\max}$ should obviously be greater than the maximum length of all constituent LFSRs. A polynomial of degree $L_{\max}$ is then recovered by our algorithm if it divides at least one polynomial of weight d and of degree at most D. We deduce from Formula (1) that the minimum possible value for D is approximatively

$$D = (d-1)!^{\frac{1}{d-1}} 2^{\frac{L_{\max}}{d-1}} \ . \tag{5}$$

This also implies that the attack can only use ciphertext blocks of length at least LC with

$$LC \geq (d-1)!^{\frac{1}{d-1}} 2^{\frac{L_{\max}}{d-1}} \ . \tag{6}$$

Moreover, we want the probability for a false alarm in the algorithm to be less than 2^{-20}. This implies that $(\sum_{k=1}^{n_c} LC(k)) - n_c D \geq \left(\frac{T}{5}\right)^2$. By replacing T by its value, we obtain the following condition

$$N_t - n_c D \geq \frac{1}{25}\left((N_t - n_c D)(2\varepsilon_{\min})^d - 3\sqrt{(N_t - n_c D)(1-(2\varepsilon_{\min})^{2d})}\right)^2$$

where $N_t = \sum_{k=1}^{n_c} LC(k)$ is the total ciphertext length. We deduce that

$$N_t - n_c D \geq \frac{\left(5 + 3\sqrt{1-(2\varepsilon_{\min})^{2d}}\right)^2}{(2\varepsilon_{\min})^{2d}} \ . \tag{7}$$

It finally follows that the total ciphertext length should satisfy

$$N_t \geq n_c (d-1)!^{\frac{1}{d-1}} 2^{\frac{L_{\max}}{d-1}} + \frac{\left(5 + 3\sqrt{1-(2\varepsilon_{\min})^{2d}}\right)^2}{(2\varepsilon_{\min})^{2d}} \ . \tag{8}$$

This value is minimal if $n_c = 1$, i.e., if all known ciphertext bits are consecutive. In this case, the minimum length of the ciphertext sequence required by the reconstruction is

$$N_t = \min_d \left[(d-1)!^{\frac{1}{d-1}} 2^{\frac{L_{\max}}{d-1}} + \frac{\left(5 + 3\sqrt{1-(2\varepsilon_{\min})^{2d}}\right)^2}{(2\varepsilon_{\min})^{2d}}\right] \ . \tag{9}$$

This formula points out that the optimal value of d increases with $L_{\max}$. Figure 2 shows how N_t and the optimal value of d vary with $\varepsilon_{\min}$, for $L_{\max} = 70$.

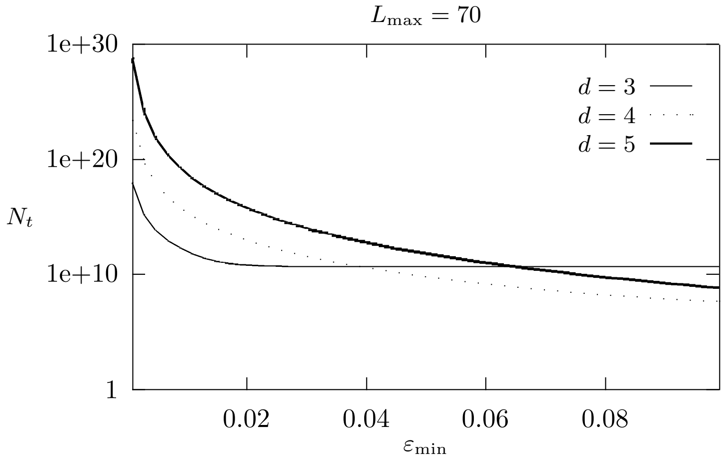


Fig. 2. Minimum ciphertext length required for $L_{\max} = 70$

In most practical situations, all ciphertext blocks have roughly the same length LC. The number n_c of such ciphertext blocks required by the reconstruction is then

$$n_c \geq \frac{\left(5 + 3\sqrt{1 - (2\varepsilon_{\min})^{2d}}\right)^2}{(2\varepsilon_{\min})^{2d}(LC - (d-1)!^{\frac{1}{d-1}} 2^{\frac{L_{\max}}{d-1}})} . \tag{10}$$

We then use the algorithm with the value of d which minimizes this formula.

The number of operations performed by the algorithm is roughly

$$\frac{D^{d-1}}{(d-1)!} d(N_t - n_c D) .$$

Using equations (5) and (8), we obtain the following complexity

$$\frac{d 2^{L_{\max}} \left(5 + 3\sqrt{1 - (2\varepsilon_{\min})^{2d}}\right)^2}{(2\varepsilon_{\min})^{2d}} .$$

Another method for recovering the feedback polynomials of the LFSRs consists in examining all polynomials of degree at most $L_{\max}$ and in computing the corresponding parity-check equations on the ciphertext sequence. A similar analysis applies to this attack. We here have to choose $D = L_{\max}$ and $d \simeq L_{\max}/2$ since the average weight of a polynomial of degree $L_{\max}$ is roughly $L_{\max}/2$. With these parameters, Formula (7) provides the minimum ciphertext length required by this second attack:

$$N'_t = L_{\max} + \frac{\left(5 + 3\sqrt{1-(2\varepsilon_{\min})^{L_{\max}}}\right)^2}{(2\varepsilon_{\min})^{L_{\max}}} .$$

We easily see that this number is much larger that the number of ciphertext bits required by our attack (see Formula (8)). Moreover, the number of operations performed by this second attack is roughly

$$\frac{d2^{L_{\max}}\left(5 + 3\sqrt{1-(2\varepsilon_{\min})^{L_{\max}}}\right)^2}{2(2\varepsilon_{\min})^{L_{\max}}} .$$

Our attack is then much more efficient than the enumeration of all polynomials of degree $L_{\max}$.

3.3 Simulation Results

We consider the following toy example of combination generator. Three LFSRs are combined by the majority function $f(x_1,x_2,x_3) = x_1x_2 + x_1x_3 + x_2x_3$. The feedback polynomials are respectively

$$\begin{aligned}
P_1(x) &= 1 + x^2 + x^4 + x^5 + x^6 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{15} \\
P_2(x) &= 1 + x^2 + x^4 + x^5 + x^6 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{14} + x^{15} + x^{17} \\
P_3(x) &= 1 + x + x^7 + x^8 + x^{10} + x^{12} + x^{15} + x^{16} + x^{17} + x^{20} + x^{21} + x^{22} + x^{23}
\end{aligned}$$

The output of this combination generator is used for encrypting a plaintext with $p_0 = 0.70$. We take $\varepsilon_{\min} = 0.1$; this value corresponds to Formula (4) with $n = 3$. We applied our algorithm with parameters $D = 3,620$ and $d = 3$ (which is the optimal value for these parameters). We used 170 ciphertext blocks of length 10,000 (i.e., around 1,200 ASCII characters). Note that Formula (10) gives $n_c = 157$. Exactly 263 trinomials have been detected by our algorithm. All of these trinomials are divisible by one of the feedback polynomials. This means that the effective probability for a false alarm is zero. Moreover, all multiples of P_1, P_2 and P_3 of degree at most 3,620 have been detected (see Table 1). This

Table 1. Detected polynomials for the toy example

	d	P_1	P_2	P_3	Total
Nb. of detected polynomials	3	208	53	2	263
Exact nb. of multiples	3	208	53	2	263

simulation required roughly one week on a DEC alpha workstation at 433 MHz.

We also checked our attack on the same example where P_1 was replaced by

$$1+x+x^3+x^6+x^7+x^8+x^{13}+x^{16}+x^{19}+x^{20}+x^{25}+x^{26}+x^{27}+x^{28}+x^{29}+x^{31}+x^{33} .$$

We here used 6,109 ciphertext blocks of length 10,000. The optimal parameters are here $D = 5,910$ and $d = 4$. For these values, all multiples of P_1, P_2 and P_3, of weight 4 and degree at most D have been detected.

4 Recovering of the Combining Function

A method for recovering the combining function was developed in [12] but it requires the knowledge of all LFSR initializations. Moreover, this technique relies on Siegenthaler's correlation attack; its complexity is then exponential in the lengths of the constituent LFSRs. We now show how to bypass these limitations and to practically reconstruct the combining function.

The number of variables of the combining function is derived from the previous step of our attack. Moreover, the previous analysis also provides an estimation of some Walsh coefficients of the combining function. Suppose that some multiples of weight d of $\prod_{i \in T} P_i$, $T \subset \{1, \ldots, n\}$, have been detected by our algorithm. For any such multiple, the mean value of the estimator Z equals $N(2p-1)^d$, where $p = P[c_t = s_t]$ with $s = g(s^1, \ldots, s^n)$ and $g(x) = 1_T \cdot x$. The values of Z obtained for all detected multiples of $\prod_{i \in T} P_i$ therefore provides an estimation of probability p. Using Formula (3), we can then compute the value of the corresponding Walsh coefficient, $\widehat{\chi_f}(1_T)$. This value is rounded to the closest multiple of 4, since all the Walsh coefficients are divisible by 4 for balanced functions.

If $\prod_{i \in T} P_i$ has degree L greater than $L_{\max}$, no multiple was detected by the algorithm. We then choose a higher value of d satisfying

$$(d-1)!^{\frac{1}{d-1}} 2^{\frac{L}{d-1}} \leq LC .$$

We then compute all multiples of $\prod_{i \in T} P_i$ of weight d and degree at most LC, and the corresponding values of Z. We deduce the involved Walsh coefficient as previously seen.

Example 2. In the toy example, the values of the estimator Z obtained for each multiple of weight 3 of P_1 provide

$$P[c_t = s_t] = 0.6003 .$$

Formula (3) gives the approximation: $\widehat{\chi_f}(1, 0, 0) = 4.01$. Similarly, we obtain the following information during the first step:

$$\widehat{\chi_f}(0, 1, 0) = \widehat{\chi_f}(1, 0, 0) = 4 \quad \widehat{\chi_f}(0, 0, 0) = 0 .$$

For each detected P_i we compute some multiples of weight 5 and of degree at most 10,000 for each product P_iP_j. Although all of these products were potentially detectable, no one was detected; we then deduce that

$$\widehat{\chi_f}(1,1,0) = \widehat{\chi_f}(1,0,1) = \widehat{\chi_f}(0,1,1) = 0 \ .$$

Similar simulations for $d = 7$ allow to find the remaining coefficient:

$$\widehat{\chi_f}(1,1,1) = -4 \ .$$

Acknowledgments. This paper is dedicated to Colonel Max Mayneris, a very enthusiastic signal officer, who very early understood the importance of working on such subjects. We will have an everlasting thought for his influence. We hope he now enjoys his retirement in his beloved Pyrénées mountains.

References

1. A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Advances in Cryptology - EUROCRYPT 2000*, Lecture Notes in Computer Science. Springer-Verlag, 2000. To appear.
2. W. Feller. *An Introduction to Probability Theory.* Wiley, 1966.
3. E. Filiol and C. Fontaine. Highly nonlinear balanced Boolean functions with a good correlation-immunity. In *Advances in Cryptology - EUROCRYPT'98*, number 1403 in Lecture Notes in Computer Science, pages 475–488. Springer-Verlag, 1998.
4. P.R. Geffe. How to protect data with ciphers that are really hard to break. *Electronics*, pages 99–101, 1973.
5. T. Herlestam. On functions of linear shift register sequences. In F. Pichler, editor, *Advances in Cryptology - EUROCRYPT '85*, number 219 in Lecture Notes in Computer Science, pages 119–129. Springer-Verlag, 1986.
6. T. Johansson and F. Jönsson. Fast correlation attacks based on turbo code techniques. In *Advances in Cryptology - CRYPTO'99*, number 1666 in Lecture Notes in Computer Science, pages 181–197. Springer-Verlag, 1999.
7. T. Johansson and F. Jönsson. Improved fast correlation attack on stream ciphers via convolutional codes. In *Advances in Cryptology - EUROCRYPT'99*, number 1592 in Lecture Notes in Computer Science, pages 347–362. Springer-Verlag, 1999.
8. D. Kahn. *The Codebreakers: The Story of Secret Writings.* Macmillan Publishing Co, 1967.
9. R. Lidl and H. Niederreiter. *Finite fields.* Cambridge University Press, 1983.
10. F.J. MacWilliams and N.J.A. Sloane. *The theory of Error-correcting codes.* North-Holland, 1977.
11. W. Meier and O. Staffelbach. Fast correlation attack on certain stream ciphers. *J. Cryptology*, pages 159–176, 1989.
12. S. Palit and B. Roy. Cryptanalysis of LFSR-encrypted codes with unknown combining function. In *ASIACRYPT'99*, number 1716 in Lecture Notes in Computer Science. Springer-Verlag, 1999.
13. R.A. Rueppel. *Analysis and Design of stream ciphers.* Springer-Verlag, 1986.
14. R.A. Rueppel and O.J. Staffelbach. Products of linear recurring sequences with maximum complexity. *IEEE Trans. Inform. Theory*, 33(1):124–131, 1987.

15. E.S. Selmer. *Linear recurrence relations over finite fields.* PhD thesis, University of Bergen, Norway, 1966.
16. T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. Inform. Theory*, IT-30(5):776–780, 1984.
17. T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Trans. Computers*, C-34(1):81–84, 1985.
18. G. Xiao and J.L. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Trans. Inform. Theory*, IT-34(3):569–571, 1988.

A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers

Vladimor V. Chepyzhov[1], Thomas Johansson[2], and Ben Smeets[3]

[1] Institute for Problems of Information Transmission, Russian Academy of Sciences, Moscow, Russia
[2] Dept. of Information Technology, Lund University, P.O. Box 118, 221 00 Lund, Sweden
[3] Ericsson Mobile Communications, 221 83 Lund, Sweden

Abstract. A new simple algorithm for fast correlation attacks on stream ciphers is presented. The advantages of the new approach are at least two. Firstly, the new algorithm significantly reduces the memory requirements compared with some recent proposals [2,3]. This allows more powerful attacks than previously. Secondly, the simplicity of the algorithm allows us to derive theoretical results. We determine the relation between the number of observed symbols, the correlation probability, and the allowed computational complexity, required for a successful attack. Hence, we can get theoretical estimates on the required computational complexity in cases when simulation is not possible.

Keywords. Stream ciphers, correlation attacks, cryptanalysis.

1 Introduction

A good stream cipher should be resistant against a *known-plaintext attack*. Here the cryptanalyst is given a plaintext and the corresponding ciphertext, and the task is to determine a key k. This is usually equivalent to the problem of finding the key k that produced a given running key $z_1, z_2, \ldots, z_N$.

The problem of cryptanalysis often involves recovering (restoring) the initial states of some linear feedback shift registers, LFSRs. As usual it is assumed that the structure of the key generator is known to the cryptanalyst.

It was noticed by Siegenthaler in [1] that it can happen that the observed output sequence (running key) is correlated to the output of a particular (target) LFSR in the generator. Thus it is reasonable to try to apply a so called divide-and-conquer attack, i.e., try to restore the initial state of the target LFSR independently of the other unknown key bits. In such a setting, one may consider the output of the target LFSR to have passed through an observation channel. The nature of such a channel may vary, but here we model the channel by the Binary (Memoryless) Symmetric Channel, BSC, with some error probability $p < 1/2$.

If $p = 1/2-\varepsilon$ then usually ε is small. In this setting an LFSR output sequence having some fixed length N can be regarded as a binary linear $[N, l]$-code, where l is the degree of the feedback polynomial of the target LFSR. The number of codewords equals the number of initial states of LFSR, that is 2^l. Thus, the cryptanalyst's problem can be reformulated as a decoding problem of this particular code in the presence of a BSC with strong noise. The problem is how to decode this linear $[N, l]$-code with as low decoding complexity as possible.

Meier and Staffelbach presented in [4] how this could be done in a very efficient way if the feedback polynomial has low weight. Essentially, they made use of iterative decoding techniques. Several minor improvements then followed [5, 6,10]. In [2,3], Johansson and Jönsson presented new ideas involving convolutional codes that improved Meier and Staffelbach's results in the case of a general feedback polynomial.

Although we are influenced by [2,3], this paper proceeds in another direction and uses the following idea. Associate with the target LFSR another binary linear (n_2, k)-code with $k < l$. The k information symbols of this code may coincide with the first k symbols of the initial state of the LFSR we want to recover. The codeword of this second code is considered to have passed through another BSC with a "double" noise level $p_2 = 2p(1-p) > p$, or $p_2 = 1/2 - 2\varepsilon^2$. As will be shown in the paper, if the length of the new code can be chosen at least $n_2 = \lceil k/C(p_2) \rceil$, then the decoding of this code leads to the recovery of the first k symbols in the initial state of the LFSR. Since the new code has dimension k, the decoding complexity is decreased from $O(2^l \times l/C(p))$ to $O(2^k \times k/C(p_2))$.

To make our method work, we need to calculate proper parity checks for construction of the second code. This is done in a precomputation step, and the result is stored on disk. In the decoding step, the symbols of the observed sequence are combined according to the parity checks, and the probability of each codeword in the code is calculated, i.e., ML-decoding. Hence, there are no memory requirements in the decoding part (as opposite to [2,3]), and thus it can be performed very efficiently.

The algorithm is most efficient when the observed output sequence is long (as for all other algorithms for fast correlation attacks), and a theoretical treatment determining the relation between the number of observed symbols, the correlation probability, and the allowed computational complexity, is given. Hence, we can get theoretical estimates of the required computational complexity in cases when simulation is not possible.

In the next section we give the model and problem definition, and in Section 3 we give some arguments on ML-decoding of linear block codes. In Section 4 and 5, the algorithm is described and a theoretical treatment is given. Section 6 contains simulation results.

2 The Cryptanalyst's Problem and Definitions

As most other authors [1]-[6], we use the approach of viewing the problem as a decoding problem. Let the target LFSR have length l and let the set of possible LFSR sequences be denoted by $\mathcal{L}$. Clearly, $|\mathcal{L}| = 2^l$ and for a fixed length N the truncated sequences from $\mathcal{L}$ form a linear $[N, l]$ block code [9], referred to as $\mathcal{C}$. Furthermore, the observed keystream sequence $\mathbf{z} = z_1, z_2, \ldots, z_N$ is regarded

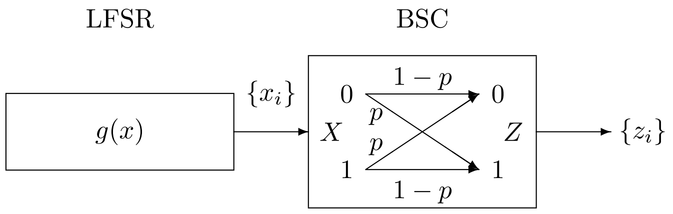


Fig. 1. The correlation attack model for initial state recovery problem.

as the received channel output and the LFSR sequence $\mathbf{x} = x_1, x_2, \ldots, x_N$ is regarded as a codeword from an $[N, l]$ linear block code. Due to the correlation between x_i and z_i, we can describe each z_i as the output of the binary symmetric channel, BSC, when x_i was transmitted. The correlation probability $1-p$, defined by

$$P(x_i = z_i) = 1 - p = 1/2 + \varepsilon,$$

gives p as the crossover probability (error probability) in the BSC. W.l.o.g we can assume $p < 0.5$. This is all shown in Figure 1. The cryptanalyst's problem can now be formulated as follows:

Statement of the problem: Let $p = 1/2 - \varepsilon < 0.5$ and let the feedback polynomial of the LFSR, denoted by $g(D)$, be of degree l. The problem is to restore the linear feedback shift register's initial state $(x_1, x_2, \ldots, x_l)$ from the observed output sequence $\mathbf{z} = (z_1, z_2, \ldots, z_N)$.

In order to derive theoretical results we use the following conjecture. The linear code under consideration is "random" enough to meet the main coding theorem: If the rate $R = k/n$ of a code is less than the capacity $C(p) = 1 - H(p)$ of the BSC then, in the ensemble of random linear (n, k) codes, the decoding error probability approaches zero. Here $H(x)$ is the binary entropy function $H(x) = -x \log_2 x - (1-x)\log_2(1-x)$. This is further described in Section 3.

Siegenthaler in [1] considered an exhaustive search through all the codewords of the above $[N, l]$-code as the decoding procedure. This algorithm is optimal because it is a maximum likelihood (ML) decoding. In [1,6] it is demonstrated that the probability of success is more than 1/2 if $N > n_0$, where n_0 is the *critical length*

$$n_0 = \lceil l/C(p) \rceil .$$

The complexity of this algorithm is about $O\left(2^l \cdot l/C(p)\right)$. The idea of fast correlation attacks is to avoid the factor 2^l and derive algorithms with complexity of order $O(2^{\alpha l})$ with respect to some $\alpha < 1$.

Let us briefly recall some results from coding theory. Since each symbol x_i is a linear combination of the l initial values we see that the set of words

$$(x_1, x_2, \ldots, x_N)$$

forms the linear code $\mathcal{C}$ and we call these words *codewords*. The word

$$(x_1, x_2, \ldots, x_l)$$

is called the information word and the symbols $x_1, x_2, \ldots, x_l$ are called information symbols. We have that

$$\begin{cases} c_l x_1 + c_{l-1} x_2 + \ldots + c_1 x_l + c_0 x_{l+1} = 0 \\ c_l x_2 + c_{l-1} x_3 + \ldots + c_1 x_{l+1} + c_0 x_{l+2} = 0 \\ \ldots \\ c_l x_{N-l} + c_{l-1} x_{N-l+1} + \ldots + c_1 x_{N-1} + c_0 x_N = 0 \end{cases}, \tag{1}$$

where c_i are coefficients of the generator polynomial $g(D) = c_0 + c_1 D + \ldots + c_l D^l$ $(c_0 = c_l = 1)$. We define the $(N-l) \times N$ matrix

$$H = \begin{pmatrix} c_l & c_{l-1} & c_{l-2} & \cdots & c_0 & 0 & \cdots & 0 \\ 0 & c_l & c_{l-1} & \cdots & c_1 & c_0 & \cdots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & c_l & c_{l-1} & \cdots & c_1 & c_0 \end{pmatrix}.$$

Thus, our code consists of all codewords such that $H\,(x_1, x_2, \ldots, x_N)^T = 0$. The matrix H is called the parity check matrix of the LFSR code. It follows from (1) that

$$\begin{cases} x_{l+1} = h_{l+1}^1 x_1 + h_{l+1}^2 x_2 + \ldots + h_{l+1}^l x_l \\ x_{l+2} = h_{l+2}^1 x_1 + h_{l+2}^2 x_2 + \ldots + h_{l+2}^l x_l \\ \vdots \\ x_i = h_i^1 x_1 + h_i^2 x_2 + \ldots + h_i^l x_l \\ \vdots \\ x_N = h_N^1 x_1 + h_N^2 x_2 + \ldots + h_N^l x_l \end{cases}. \tag{2}$$

If we denote $h_i(D) = h_i^1 + h_i^2 D + \ldots + h_i^l D^{l-1}$, then clearly

$$h_i(D) = D^{i-1} \bmod g(D) \text{ for } i = 1, 2, \ldots, N.$$

Therefore the code $\mathcal{C}$ has the $(l \times N)$-generator matrix

$$G = \begin{pmatrix} h_1^1 & h_2^1 & \cdots & h_N^1 \\ h_1^2 & h_2^2 & \cdots & h_N^2 \\ \vdots & & \cdots & \\ h_1^l & h_2^l & \cdots & h_N^l \end{pmatrix}. \tag{3}$$

The observed output sequence, which we call the received word and denote

$$\mathbf{z} = (z_1, z_2, \ldots, z_N),$$

is regarded as a very noisy version of the unknown codeword $(x_1, x_2, \ldots, x_N)$. The initial problem can now be reformulated as a decoding problem:

Restatement of the problem: Let $p < 0.5$ and let the generator polynomial, denoted by $g(D)$, be of degree l. Consider the corresponding $[N, l]$ code $\mathcal{C}$.

The problem is to determine the transmitted codeword from the received word $\mathbf{z}$.

Remark. In the theory of LFSR's and stream ciphers, the typical values for p are closed to 1/2, say, $p = 0.25$, 0.3, 0.4, 0.45. However in the theory of error-correcting codes the typical values of p are much smaller, say, for example, $p = 0.1,\ 0.05$. So we have to cope with a much stronger noise than is usual in coding theory. On the other hand, unlike in error-correcting codes, it is sufficient to demonstrate that our algorithm is able to recover the initial state with some nonzero probability, say 1/2. Thus, a much higher error-rate in decoding can be accepted.

3 ML-Decoding of Linear Codes

In this section we review ML-decoding for binary linear codes and their error-correcting capability. The best selection for the information word is obtained by

Maximum Likelihood decoding (ML-decoding):
Let $\mathcal{C} = \{\mathbf{x}\}$ be an (n, k)-code and let $\bar{\mathbf{x}}$ be the transmitted codeword and let $\mathbf{z}$ be the received word. Now put

$$\hat{\mathbf{x}}_0 \overset{def}{=} \arg\min_{\mathbf{x}\in\mathcal{C}} \operatorname{dist}(\mathbf{x}, \mathbf{z}).$$

Here $\operatorname{dist}(\mathbf{x}, \mathbf{y})$ denotes the Hamming distance between $\mathbf{x}$ and $\mathbf{y}$, i.e., the number of ones in the binary vector $\mathbf{x} + \mathbf{y}$. Notice that the previously described exhaustive search algorithm is exactly ML-decoding. Furthermore, let

$$P_e(p) \overset{def}{=} \Pr(\hat{\mathbf{x}}_0 \neq \bar{\mathbf{x}}) \quad \text{-error probability of ML-decoding.}$$

Note that $P_e(p)$ does not depend on $\bar{\mathbf{x}}$ because the code $\mathcal{C}$ is linear. It is known that ML-decoding has the smallest error probability among all decoding algorithms. So it is optimal. Using a random coding argument, Gallager, Berlekamp and Shannon proved the following.

Theorem 1 ([8]). *Let $C = 1 - H(p)$ denote the capacity of the transmission (observation) channel and let the transmission rate $R = k/n$ satisfy $R < C$. Then*

$$E\left[P_e(p)\right] \leq 2^{-\tau(R)n}, \quad \tau(R) = \tau(R, p),$$

where $E\left[P_e(p)\right]$ is the mathematical expectation of the random value $P_e(p)$ in the ensemble of random linear (n, k)-codes. $\tau(R)$ is called the random coding exponent, and $\tau(R) > 0$ for all $R < C$.

Thus we can apply the ML-decoding to the code $\mathcal{C}$ with length N satisfying the inequality $l/N < C(p)$, that is $N > l/C(p)$.

Recall that $p = 1/2 - \varepsilon$. A useful approximation of $C(p)$ is

$$C(p) \approx \varepsilon^2 \cdot 2/\left(\ln 2\right). \tag{4}$$

Simulations show that the critical length $N = n_0 \approx 0.35 \cdot l \cdot \varepsilon^{-2}$ provides the probability of successful decoding close to 1/2, while for $N = 2n_0$ the probability is close to 1 (see [1]). Recall that the complexity of this algorithm has order $O(2^l \cdot l \cdot \varepsilon^{-2})$, so it is very time consuming. Yet we can turn this into something useful as we will see in the next section.

4 Description of a New Fast Correlation Attack

Let $k < l$ be fixed. We shall describe a procedure that recovers the symbols $x_1, x_2, \ldots, x_k$ when observing $z_1, z_2, \ldots, z_N$. In the system (2) we look for pairs of equations such that,

$$h_i^{k+1} = h_j^{k+1},\ h_i^{k+2} = h_j^{k+2}, \ldots, h_i^l = h_j^l,\ 1 \le i \ne j \le N. \tag{5}$$

We find all such pairs of equations. Let the number of such distinct pairs be n_2. Denote the indices of all such pairs as $\{i_1, j_1\}, \{i_2, j_2\}, \ldots, \{i_{n_2}, j_{n_2}\}$.

If i and j satisfies (5), then the sum $x_i + x_j$ is a linear combination of information symbols $x_1, x_2, \ldots, x_k$ only, and is independent of the remaining information symbols $x_{k+1}, x_{k+2}, \ldots, x_l$,

$$x_i + x_j = \left(h_i^1 + h_j^1\right) x_1 + \left(h_i^2 + h_j^2\right) x_2 + \ldots + \left(h_i^k + h_j^k\right) x_k. \tag{6}$$

This means that the sequence

$$(X_1, X_2, \ldots X_{n_2}) = (x_{i_1} + x_{j_1}, x_{i_2} + x_{j_2}, \ldots, x_{i_{n_2}} + x_{j_{n_2}})$$

forms an (n_2, k)-code, referred to as $\mathcal{C}_2$, whose information symbols are $(x_1, x_2, \ldots, x_k)$, i.e., it has dimension k. The generator matrix of the (n_2, k)-code $\mathcal{C}_2$ is

$$G_2 = \begin{pmatrix} h_{i_1}^1 + h_{j_1}^1 & h_{i_2}^1 + h_{j_2}^1 & \cdots & h_{i_{n_2}}^1 + h_{j_{n_2}}^1 \\ h_{i_1}^2 + h_{j_1}^2 & h_{i_2}^2 + h_{j_2}^2 & \cdots & h_{i_{n_2}}^2 + h_{j_{n_2}}^2 \\ \vdots & & \cdots & \\ h_{i_1}^k + h_{j_1}^k & h_{i_2}^k + h_{j_2}^k & \cdots & h_{i_{n_2}}^k + h_{j_{n_2}}^k \end{pmatrix}. \tag{7}$$

We denote

$$Z_1 = z_{i_1} + z_{j_1}, Z_2 = z_{i_2} + z_{j_2}, \ldots, Z_n = z_{i_{n_2}} + z_{j_{n_2}}. \tag{8}$$

Since we observe the output symbols $(z_{i_1}, z_{j_1}, z_{i_2}, z_{j_2}, \ldots, z_{i_{n_2}}, z_{j_{n_2}})$ we can calculate also $(Z_1, Z_2, \ldots, Z_{n_2})$, that is, a word acting as a received word for $\mathcal{C}_2$. If $(e_1, e_2, \ldots, e_N)$ is the noise sequence of the code $\mathcal{C}$, $(e_i = x_i + y_i)$, then clearly the noise sequence $(E_1, E_2, \ldots, E_{n_2})$ for $\mathcal{C}_2$ is

$$E_1 = e_{i_1} + e_{j_1}, E_2 = e_{i_2} + e_{j_2}, \ldots, E_{n_2} = e_{i_{n_2}} + e_{j_{n_2}}.$$

Since our model is the BSC, all the e_i's are independent random binary random variables with error probability p. It is then clear that $E_m = e_{i_m} + e_{j_m}$ for

$i_m \neq j_m$ are also independent binary random variables for all $1 \leq m \leq n_2$ with error probability

$$p_2 = \Pr(e_{i_m} + e_{j_m} = 1) = 2p(1-p) = 1/2 - 2\varepsilon^2.$$

Thus, we have created a new code $\mathcal{C}_2$ with smaller dimension but the BSC over which the codeword is transmitted has a stronger noise p_2, i.e., $p_2 > p$. To restore the symbols $x_1, x_2, \ldots, x_k$ we have to decode the $[n_2, k]$-code C_2 in the BSC with the stronger noise p_2. However, as long as the length of the new code $\mathcal{C}_2$ guarantees unique decoding, this new code will be decoded significantly faster than the LFSR code $\mathcal{C}$.

As before, we apply a simple ML-decoding procedure when decoding $\mathcal{C}_2$. By Theorem 1 we need a code length larger than the critical length n_0, in this case

$$n_2 > k/C(p_2), \tag{9}$$

to get the reliable recovery of the information word. We now are ready to describe the algorithm for the recovery of the initial state.

Algorithm A1.

Data: the length l of target LFSR, and the generator polynomial $g(D)$.

Precomputation.

Fix a computational complexity level by choosing a $k < l$ (for example. $k = l/2$). Construct the generator matrix G (see (3)). Using a sorting algorithm, sort the columns of G with respect to

$$\left(h_i^{k+1}, h_i^{k+2}, \ldots, h_i^l\right),\ i = 1, 2, \ldots, N.$$

Find all pairs $\{i, j\}$ that satisfy (5). For each pair, store the indices i, j together with the value of $\left(h_i^1 + h_j^1, h_i^2 + h_j^2, \ldots, h_i^k + h_j^k\right)$. Hence, we have constructed the code $\mathcal{C}_2$ with generator matrix G_2 (see (7)).

Decoding.

Input: The received (observed) vector $(z_1, z_2, \ldots, z_N)$.

Step 1. Compute $(Z_1, Z_2, \ldots, Z_{n_2})$ (see (8)).

Step 2. Decode the code $\mathcal{C}_2$ with the generator matrix (7) using exhaustive search through all the 2^k codewords of $\mathcal{C}_2$, and select the information word $(x_1, x_2, \ldots, x_k)$ with highest probability.

Remark 1. After having restored $(x_1, x_2, \ldots, x_k)$ we need to restore the remaining part of the initial state, $(x_{k+1}, x_{k+2}, \ldots, x_l)$. An obvious way would be to repeat the proposed procedure for some other information bits, say $(x_{k+1}, x_{k+2}, \ldots, x_{2k})$. We can use the same parity checks, since the code is cyclic.

However, with knowledge of the first k information symbols, the remaining problem is much simplified compared to the original problem. Hence we can discard the complexity and the error probability of this step. (E.g. if we restore the 20 first information bits of a length 80 LFSR, we use the obtained values of these 20 first information bits and get a new decoding problem but now only for a length 60 LFSR.)

It is clear that we do not have to restrict ourselves to finding pairs of parity check equations of the form (6), but can consider triples, etc. We describe the general form of the algorithm, that uses sets of t parity check equations whose sums only include the information symbols $(x_1, x_2, \ldots, x_k)$.

Algorithm A2.

Data: the length l of target LFSR and the generator polynomial $g(D)$.

Precomputation.

Fix a computational complexity level by choosing a $k < l$ and a $t \geq 2$. Construct the generator matrix G (see (3)). Sort all columns of G with respect to $\left(h_i^{k+1}, h_i^{k+2}, \ldots, h_i^l\right),\ i = 1, \ldots, N$.

Then find all sets of t indices $\{i(1), i(2), \ldots, i(t)\}$ that satisfy

$$\sum_{j=1}^{t} h_{i(j)}^m = 0, \text{ for } m = k+1, k+2, \ldots, l.$$

Let the number of such sets be n_t. For each set, store the indices $i(1), i(2), \ldots, i(t)$ together with the value of

$$\left(\sum_{j=1}^{t} h_{i(j)}^1, \sum_{j=1}^{t} h_{i(j)}^2, \ldots, \sum_{j=1}^{t} h_{i(j)}^k\right).$$

Hence, we have constructed an (n_t, k)-code $\mathcal{C}_t$.

Decoding.

Input: The received (observed) vector $(z_1, z_2, \ldots, z_N)$.

Step 1. Compute

$$(Z_1 = \sum_{j=1}^{t} z_{i_1(j)}, Z_2 = \sum_{j=1}^{t} z_{i_2(j)}, \ldots, Z_n = \sum_{j=1}^{t} z_{i_n(j)}).$$

Step 2. Decode the code $\mathcal{C}_t$ using exhaustive search through the all 2^k code words of $\mathcal{C}_t$ and output $(x_1, x_2, \ldots, x_k)$.

Using our model of a BSC and assuming that all the e_i's are the independent random binary values with probability $p = 1/2 - \varepsilon$, it can be shown that

$$E_m = \sum_{j=1}^{t} e_{i_m(j)}$$

are independent random binary variables with error probability

$$p_t = \Pr(\sum_{j=1}^{t} e_{i_m(j)} = 1) = 1/2 - 2^{t-1}\varepsilon^t.$$

We will study this algorithm further in the next section.

5 A Theoretical Analysis of the Proposed Algorithm

We consider theoretical results for algorithm A1 in more detail. We later transfer the results to algorithm A2.

The following lemma shows how many pairs of indices i and j, $1 \leq i, j \leq N$, satisfying the conditions (5), we can expect to find, and how this expectation number n_2 depends on N and k.

Lemma 1 (Birthday paradox). *Let $\xi_1, \xi_2, \ldots, \xi_N$ be random variable that are uniformly distributed on the set of L values. We assume that all this variables are pairwise independent. Denote by ψ the number of pairs $\{i, j\}$ such that $\xi_i = \xi_j$. Then*

$$E(\psi) = \frac{N(N-1)}{2L},$$

where $E(\psi)$ is the mathematical expectation of ψ.

Proof. For every pair $\{i, j\}, i < j$ we denote the random value

$$\pi_{i,j} = \begin{cases} 1 \text{ if } \xi_i = \xi_j \\ 0 \text{ otherwise} \end{cases}$$

The number of these values is $\frac{N(N-1)}{2}$. Since the values ξ_i and ξ_j are independent it follows easily that $\Pr(\pi_{i,j} = 1) = L^{-1}$. Hence $E(\pi_{i,j}) = L^{-1}$. It is clear that

$$\psi = \sum_{\{i,j\}} \pi_{i,j}$$

and therefore

$$E(\psi) = \sum_{\{i,j\}} E(\pi_{i,j}) = \frac{N(N-1)}{2} L^{-1}.$$

□

To apply this lemma we set

$$\xi_i = \left(h_i^{k+1}, h_i^{k+2}, \ldots, h_i^l\right), \; i = l+1, \ldots, l+N.$$

Therefore $L = 2^{l-k}$. From the theory of LFSRs it follows that ξ_i is a good generator of quasi random values with uniform distribution that are pairwise independent.

Corollary 1. *The number n_2 of pairs $\{i, j\}$ that satisfy (5) has expectation*

$$E(n_2) = \frac{N(N-1)}{2} 2^{-(l-k)}. \tag{10}$$

Simulations show that for particular LFSRs the number n_2 is closed to $E(n_2)$ in the formula (10) (see the next section).

Combining (9) and (10) we obtain the length N of the output of the LFSR generator that we have to observe in order to recover the symbols $(x_1, x_2, \ldots, x_k)$ with high probability (close to 1/2).

Theorem 2. *With given k, l, ε, the required length N of the observed sequence $\mathbf{z}$ for algorithm A1 to succeed is*

$$N \approx 1/2 \cdot \sqrt{k \cdot (\ln 2)} \cdot \varepsilon^{-2} \cdot 2^{\frac{l-k}{2}}. \tag{11}$$

Proof. (Sketch) By Corollary 1 we can expect the length n_2 of the code $\mathcal{C}_2$ to be roughly $\frac{N(N-1)}{2} 2^{-(l-k)}$. We know from (9) that it is sufficient for n_2 to be at least $k/C(p_2)$ for a high probability of successful decoding. Using the approximation

$$C(p_2) \approx (2\varepsilon^2)^2 \cdot \frac{2}{\ln 2},$$

and the approximation $N(N-1) \approx N^2$ we end up with the expression (11). □

Theorem 2 characterizes the proposed algorithm in a theoretical way. It describes the relation between the number of observed symbols, the correlation probability, and the allowed computational complexity, required for a successful attack. We have described k as the level of computational complexity. Let us now look closer at the exact complexity for a given value of k.

The computational complexity is divided into two parts, one precomputation part and one decoding part. In precomputation, the calculation of all parity checks for the $\mathcal{C}_2$ code is of order $O(N \log N)$. We also need to store the generator matrix G_2 together with the two index positions creating each column in G_2. The storage requirement is at most $n_2(k + 2\log_2 N)$ bits.

The complexity of the decoding step is given as follows.

Corollary 2. *The decoding complexity in algorithm A1 is of the order*

$$2^k \cdot k \cdot \frac{\log_2 2}{8\epsilon^4}.$$

Proof. We run through 2^k codewords with length n_2, where $n_2 \approx k \cdot \frac{\log_2 2}{8\epsilon^4}$.

So by taking a small k, we reduce the 2^k factor but pay in the growth of n_2 and thus also in the length of the observed sequence N.

Important to note is that the decoding part has essentially no memory requirements, since we only keep the most probable information word in memory (however, using some memory can speed up the decoding further). This is in contrast to the algorithms in [2,3], where an extensive amount of memory is used in the decoding part. In fact, it is stated in [3] that the main bottleneck is the memory requirements, and reducing it enables more powerful attacks by the fact that higher computational complexity can be allowed (in [2] they could in some particular cases only use a few hours of computing time, since trying to use more time required too much memory).

Let us now consider algorithm A2. A similar reasoning as above will provide us with the following theorem, similar to Theorem 2.

Theorem 3. *With given k, l, ε, t, the required length N of the observed sequence $\mathbf{z}$ for algorithm A2 to succeed is*

$$N \approx 1/4 \cdot (2kt! \ln 2)^{1/t} \cdot \varepsilon^{-2} \cdot 2^{\frac{l-k}{t}}, \tag{12}$$

assuming $N >> n_t$.

Proof. (Sketch) We can expect the length of the code $\mathcal{C}_t$ to be approximately $\frac{N^t}{t!} 2^{-(l-k)}$. The length should be at least $k/C(p_t)$ for a high probability of successful decoding. Using the approximation

$$C(p_t) \approx (2^{t-1}\varepsilon^t)^2 \cdot \frac{2}{\ln 2},$$

we end up with the expression (12).

The complexity of the decoding step is easily calculated.

Corollary 3. *The decoding complexity for algorithm A2 is of the order*

$$2^k \cdot k \cdot \frac{2 \log_2 2}{(2\epsilon)^{2t}}.$$

Proof. We run through 2^k codewords with length n_t, where $n_t \approx k \cdot \frac{2 \log_2 2}{(2\epsilon)^{2t}}$.

As will be apparent from calculated examples, algorithm A2 is in general more powerful than A1, i.e., using not pairs but sums of three or four columns is more powerful. However, the complexity and storage requirements in the precomputation part is increasing. The calculation of all parity checks for the $\mathcal{C}_t$ code is now of order $O(N^2)$ or higher. The length n_t of G_t in algorithm A2 has increased compared to n_2 in A1. The storage is now in the order of $n_t(k + t \log_2 N)$ bits.

Finally, we must note that for algorithm A2, it can happened that $N >> n_t$ is not true. This will mean that some column positions of G will be used several times when constructing $\mathcal{C}_t$. Hence, the assumption of independence between positions in $\mathcal{C}_t$ is no longer true. The algorithm still performs well but the performance will be slightly worse than stated in Theorem 3 (due to the dependence).

6 Simulations Results

To check the performance as well as the correctness of our assumptions/conjectures, we have made extensive simulations. The simulations were done on a Sun Sparc Ultra-80 computer running under Solaris.

First, we compare the length of the code $\mathcal{C}_2$ with the expected value $E(n_2)$. We consider a random primitive polynomial $g(D)$ of the degree $l = 60$. We set

$k = 20$. The following table reflects the precomputation for algorithm A1.

N	$E(n_2)$	n_2 (we found)	p_0
$3 \cdot 10^7$	409	445	0.25
$5 \cdot 10^7$	1137	1138	0.3
10^8	4547	4567	0.35
$2 \cdot 10^8$	18190	18404	0.4
$3.5 \cdot 10^8$	55707	56142	0.43

Here p_0 is the critical value of the error probability of BSC such that $k/n_2 = C(2p_0(1-p_0))$ when $k = 20$. Notice that the actual values of n_2 are close to the theoretical expected values $E(n_2)$.

The next table shows the probability of decoding error for algorithm A1, depending on the error probability p. This can be compared with the theoretical results in Theorem 2.

(i) $N = 5 \cdot 10^7,\ n_2 = 1138,\ p_0 = 0.3$

p	0.29	**0.3**	0.31	0.33
$P_e(A1)$	0.2	0.3	0.5	0.8

(ii) $N = 3.5 \cdot 10^8,\ n_2 = 56142,\ p_0 = 0.43$

p	0.41	0.42	**0.43**	0.44
$P_e(A1)$	0.01	0.1	0.6	0.9

The following tables show how the parameter k influences the required length N of the observed sequence as well as the time complexity of the decoding algorithm for different p. As before, we choose $l = 60$ and $t = 2$.

(i) $p = 0.3$

k	N	n_2	$P_e(A1)$	Decoding time
20	$5 \cdot 10^7$	1138	0.3	1.5 sec
23	$1.85 \cdot 10^7$	1281	0.4	12 sec
25	$9.7 \cdot 10^6$	1472	0.3	1 min
30	$2 \cdot 10^6$	1800	0.1	30 min

(ii) $p = 0.4$

k	N	n_2	$P_e(A1)$	Decoding time
20	$2 \cdot 10^8$	18404	0.4	20 sec
23	$7.38 \cdot 10^7$	19561	0.5	3 min
25	$3.86 \cdot 10^7$	21329	0.6	14 min
30	$7.45 \cdot 10^6$	25980	0.5	9 h

Notice that all the precomputation, i.e., finding all the corresponding pairs of checks for all k and N in the above table, was completed in approximately 2 hours.

We now consider simulation results for the same polynomial of degree $l = 60$ but with $t = 3$. Here n_3 stands for the number of triples of checks we found for the corresponding parameters k and N. The values of n_3 are close to their theoretical expectations.

(i) $p = 0.3$

k	N	n_3	$P_e(A2)$	Decoding time
20	$3.55 \cdot 10^5$	6670	0.4	8 sec
23	$1.86 \cdot 10^5$	7633	0.4	1 min 20 sec
25	$1.21 \cdot 10^5$	8433	0.3	5 min
28	$6.3 \cdot 10^4$	9466	0.6	1 h

(ii) $p = 0.4$

k	N	n_3	$P_e(A2)$	Decoding time
20	$1.42 \cdot 10^6$	433451	0.5	10 min
23	$7.44 \cdot 10^5$	512108	0.3	1 h 30 min
24	$6 \cdot 10^5$	523734	0.6	4 h
28	$2.5 \cdot 10^5$			

In the last table, the result for $k = 28$ is missing because for $N = 2.5 \cdot 10^5$ the expected number of checks $n_3 \approx 6 \cdot 10^5$ is larger than N. In this case the sums of the corresponding triples of the received sequence are no longer independent and the model of the memoryless BSC does not apply (See the condition of Theorem 3). If we use parameters k for which $n_3(N, k) > N$, we can not expect the performance to follow Theorem 3, although it can still be good.

It is worth also to note that the precomputation time for $t = 3$ grows as N^2. For example, for $k = 20$ and $N = 1.42 \cdot 10^6$ the computer worked 4 days to find all 433451 triples of checks. This means that the precomputation becomes more time consuming when increasing t from $t = 2$ to $t = 3$.

It is interesting to compare the results for $t = 2$ and for $t = 3$. We can see that for equal decoding complexities, the length N can be reduced significantly for $t = 3$. Compare, for example, for $p = 0.3$ the row $k = 23$ ($t = 2$) and the row $k = 20$ ($t = 3$). The reduction factor is 50. The factor is 30 for $p = 0.4$; see the row $k = 25$ ($t = 2$) and the row $k = 20$ ($t = 3$). The price for this reduction is the increase in precomputation complexity.

Finally, we have simulated a set of attacks on LFSRs with $l = 70$ and $p = 0.35$ for different lengths N, and measured the total CPU time for the attacks. The tables below show the result when the parameter k is varying.

(i) $t = 2,\ p = 0.35$

k	N	n_2	$P_e(A1)$	Decoding time
25	$5.48 \cdot 10^8$	4270	0.4	3 min
27	$2.85 \cdot 10^8$	4631	0.5	13 min
28	$2.05 \cdot 10^8$	4944	0.3	30 min
30	$1.06 \cdot 10^8$	5144	0.5	2 h 20 min

(ii) $t = 3,\ p = 0.35$

k	N	n_3	$P_e(A2)$	Decoding time
25	$2.16 \cdot 10^6$	47716	0.5	40 min
26	$1.74 \cdot 10^6$	49860	0.4	1 h 30 min
28	$1.12 \cdot 10^6$	53482	0.4	6 h 20 min

For $t = 2$ the largest precomputation complexity ($k = 25$) is 6 hours. For $t = 3$ the largest precomputation complexity ($k = 25$) is 12 days. Moreover, we were able to reach the value $p = 0.4$ for $t = 2$ observing $4.61 \cdot 10^8$ symbols ($k = 28, n_2 = 24160$). The decoding time is 2 hours. The precomputation time is 10 hours. The same can be done for $t = 3$ observing $4.85 \cdot 10^6$ symbols ($k = 25, E(n_2) = 540414$). The decoding time is 5 hours. The precomputation time will be 2 months.

It should finally be noted that it is easy to get very good estimates on the complexity in other cases by using the derived theoretical results and the simulated values above.

We believe that with a set of parallel PC:s and a few weeks of computation one could use $30 \leq k \leq 35$ and restore LFSRs of length 80-100 using algorithm A2 with $t = 3, 4$.

7 Conclusions

We have demonstrated a new algorithm for fast correlation attacks on stream ciphers. The new algorithm significantly reduces the memory requirements compared with some recent proposals [2,3]. Also, we could derive the relation between the number of observed symbols, the correlation probability, and the allowed computational complexity, required for a successful attack.

Since the algorithm can be very efficiently implemented, the performance (highest error probability for given computational complexity) is better than the algorithms in [2,3]. The performance depends on the computational complexity, and it is not always easy to do a fair comparison between two different algorithms. But the actual simulations that we have done have proved this algorithm to be the fastest.

In conclusion, we think that the simplicity of the proposed algorithm is a great advantage and that this can contribute further progress in the area.

References

1. T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only", *IEEE Trans. Comput.*, Vol. C-34, pp. 81-85, 1985.
2. T. Johansson, F. Jönsson, "Improved fast correlation attacks on stream ciphers via convolutional codes", *Proceedings of EUROCRYPT'99*, Springer-Verlag, LNCS 1592, pp. 347-362.
3. T. Johansson, F. Jönsson, "Improved fast correlation attacks on stream ciphers via convolutional codes", *Proceedings of CRYPTO'99*, Springer-Verlag, LNCS 1666, pp. 181-197.

4. W. Meier, and O. Staffelbach, “Fast correlation attacks on certain stream ciphers”, *J. Cryptology*, pp. 159-176, 1989.
5. M. Mihaljevic, and J.Dj. Golić, “A fast iterative algorithm for a shift register initial state reconstruction given the noisy output sequence”, *Proc. Auscrypt'90*, Springer-Verlag, LNCS 453, Eds. J.Seberry and J. Pieprzyk, pp. 165-175, 1990.
6. V. Chepyzhov, and B. Smeets, “On a fast correlation attack on stream ciphers”, *Adv. Crypt.-EUROCRYPT'91*, Brighton, UK, Springer-Verlag, LNCS 547, Ed, D.W. Davies, pp. 176-185, 1991.
7. J.Dj. Golić, “Computation of low-weight parity-check polynomials”, *Electronic Letters*, Vol.32, No. 21, Oct., pp. 1981-1982, 1996.
8. R.G. Gallager, *Information Theory and Reliable Communications*, John Wiley and Sons, Inc. New York, London, Sydney, Toronto, 1968.
9. F. MacWilliams, N. Sloane, *The theory of error correcting codes*, North Holland, 1977.
10. W. Penzhorn, “Correlation attacks on stream ciphers: Computing low weight parity checks based on error correcting codes”, FSE'96, Springer-Verlag, LNCS 1039, pp. 159–172.

A Low-Complexity and High-Performance Algorithm for the Fast Correlation Attack*

Miodrag J. Mihaljević[1], Marc P.C. Fossorier[2], and Hideki Imai[3]

[1] Mathematical Institute, Serbian Academy of Science and Arts, Kneza Mihaila 35, 11001 Belgrade, Yugoslavia
email: `miodragm@turing.mi.sanu.ac.yu`

[2] Department of Electrical Engineering, University of Hawaii, 2540 Dole St., Holmes Hall 483, Honolulu, HI 96822, USA
email: `marc@spectra.eng.hawaii.edu`

[3] University of Tokyo, Institute of Industrial Science, 7-22-1, Roppongi, Minato-ku, Tokyo 106-8558, Japan
email: `imai@iis.u-tokyo.ac.jp`

Abstract. An algorithm for cryptanalysis of certain keystream generators is proposed. The developed algorithm has the following two advantages over other reported ones: (i) it is more powerful and (ii) it provides a high-speed software implementation, as well as a simple hardware one, suitable for high parallel architectures. The novel algorithm is a method for the fast correlation attack with significantly better performance than other reported methods, assuming a lower complexity and the same inputs. The algorithm is based on decoding procedures of the corresponding binary block code with novel constructions of the parity-checks, and the following two decoding approaches are employed: the a posterior probability based threshold decoding and the belief propagation based bit-flipping iterative decoding. These decoding procedures offer good trade-offs between the required sample length, overall complexity and performance. The novel algorithm is compared with recently proposed improved fast correlation attacks based on convolutional codes and turbo decoding. The underlying principles, performance and complexity are compared, and the gain obtained with the novel approach is pointed out.

Keywords: stream ciphers, keystream generators, linear feedback shift registers, fast correlation attack, decoding.

1 Introduction

An important method for attack or security examination of certain stream ciphers based on nonlinear combination keystream generators composed of several linear feedback shift registers (LFSR's) (see [11], for example) are: basic correlation attack [18], and particularly the fast correlation attacks considered in a

* This work was supported by JSPS Grant RFTF 96P00604 and NSF Grant CCR-97-32959

number of papers, including [12], [20], [13], [14], [2], [3], [8], [9] and [6]. Developing or improving techniques for realization of the fast correlation attack is a standard cryptologycal problem.

The basic ideas of all reported fast correlation attacks include the following two main steps:
- Transform the cryptographic problem into a suitable decoding one;
- Apply (devise) an appropriate decoding algorithm.

There are two main approaches for realization of the fast correlation attack. The first one is based on decoding techniques for block codes (introduced in [12] and [19]), and the second one is based on decoding techniques for convolutional codes (recently proposed in [8] and [9]).

The main underlying ideas for the fast correlation attacks based on linear binary block codes decoding is the iterative decoding principle introduced in [4]. For example, all the fast correlation attacks reported in [12], [19], [13], [20], [14], [2], and [3], could be considered as variants of iterative decoding based on simple bit-flipping (BF) [4] or iterative extensions of *a posterior probability* (APP) decoding [10]. Most of these methods (practically all except the method from [13]) are restricted on the LFSR feedback polynomials of low weight. Due to the established advantages of belief propagation (BP) based iterative decoding over iterative APP (see [5], for example), the application of BP based iterative decoding for realization of the fast correlation attack has been recently reported in [6]. The main goal of [6] was to report the potential gain and its origins when BP based iterative decoding is employed instead of APP based decoding, assuming the same construction method of the parity-checks and the same overall structure of the algorithm for fast correlation attack. A comparison of the iterative decoding approaches based on simple, APP and BP based decodings for the fast correlation attack is reported in [15].

New methods for fast correlation attack based on the theory of convolutional codes are given in [8]-[9]. They can be applied to arbitrary LFSR feedback polynomials, in opposite to the previous methods, which mainly focus on feedback polynomials of low weight. The proposed algorithm transforms a part of the code **C** steaming from the LFSR sequence into a convolutional code, based on finding suitable parity check equations for **C**. The approach considers a decoding algorithm that includes memory, but still has a low decoding complexity. With respect to the previous methods, this allows looser restrictions on the parity check equations that can be used, leading to many more equations. As the final decoding method, the Viterbi algorithm with memory orders of 10-15 was used. The results reported in [8] improve significantly the few previous results for high weight feedback polynomials, and are in many cases comparable with that corresponding to low weight feedback polynomials. Further developments of the idea for fast correlation attack based on decoding of certain convolutional codes are presented in [9] where new methods employing the techniques used for constructing and decoding turbo codes are proposed. The most powerful technique presented in [9] is based on the turbo decoding approach with M component convolutional codes and iterative APP decoding employing the BCJR algorithm [1] (see also [7]).

Recent interests and the advances in developing algorithms for the fast correlation attack have raised a natural question of further improvements of the fast correlation attack, especially in the light of fast implementations.

The main goal of this paper is to propose an algorithm for the fast correlation attack suitable for a high-speed software implementation, as well as for a simple hardware one. Most existing algorithms can be considered as inappropriate ones for this goal assuming an LFSR feedback polynomial of arbitrary weight. Accordingly, our intention was to develop an algorithm which employs *mod*2 additions and simple logical operations for processing, so that it is suitable for highly parallel architectures and high speed software or hardware implementations. Also, our goal is to propose an algorithm which yields possibility for trade-offs between length of the required sample, overall complexity and performance.

In this paper, a more powerful algorithm for the fast correlation attack with significantly better performance (which does not depend on the LFSR feedback polynomial), assuming the same inputs and lower complexity than other reported methods, is proposed. The proposed algorithm is based on a novel method for constructing the parity-checks, motivated by the approach of [8] and [9], and two decoding approaches of the corresponding binary block code, APP threshold decoding [10] and iterative decoding employing BP-like BF (see [4]). The construction of the parity-checks is based on searching for certain parity-check equations and theirs linear combinations employing the finite-state machine model of an LFSR with primitive characteristic polynomial. The expected numbers of parity-checks per parity bit are derived, showing that a large number of appropriate parity-checks can be constructed. An analysis of the algorithm performance and complexity is presented. The novel algorithm is compared with recently proposed improved fast correlation attacks based on convolutional codes and turbo decoding. The underlying principles, performances and complexities are compared, and the gains obtained with the novel approach are pointed out. It is shown that assuming the same input, the novel algorithm yields better performance and lower complexity than the best algorithm reported up-to-now.

The paper is organized as follows. Section 2 presents preliminaries. Section 3 points out the main underlying results for the construction of a novel algorithm for the fast correlation attack. Complete specification of the proposed algorithm is given in Section 4. Experimental analysis of the performance is presented in Section 5, as well as a discussion of the complexity issue. Comparisons between the recently reported improved fast correlation attacks, and the proposed algorithm are given in Section 6. Finally, the results of this paper are summarized in Section 7.

2 Decoding Concept for the Fast Correlation Attack

Recall that, the correlation means that the mod 2 sum of corresponding outputs of the LFSR and the generator can be considered as a realization of a binary random variable which takes value 0 and 1 with the probabilities $1-p$ and p, respectively, $p \neq 0.5$.

The fast correlation attack on a particular LFSR, with primitive feedback polynomial, in a nonlinear combining generator given the segment of the generator output can be considered as follows:

- The n-bit segment of the output sequence from the length-k LSFR is a codeword of an (n,k) punctured simplex code;
- The corresponding n-bit segment of the nonlinear combination generator output is the corresponding noisy codeword obtained through a BSC with crossover probability p;
- The problem of the LFSR initial state reconstruction, assuming known characteristic polynomial, is equivalent to the problem of decoding after transmission over a BSC with crossover probability p.

The decoding approach employed in this paper is based on combination of a restricted exhaustive search over a set of hypotheses and a one-step or an iterative decoding technique. The exhaustive search is employed in order to provide a possibility for construction of suitable parity-check equations relevant for high performance of complete decoding. This approach could be considered as a particular combination of the minimum distance decoding and another decoding technique.

Recall that a parity-check equation which involves a smaller number of bits is more powerful than a higher weight one. Also note that performance associated with a set of the parity-checks depends on its cardinality as well as on the parity-check weight distribution. Finally, the overall complexity of a decoding procedure depends on the number and weights of the employed parity-checks. Accordingly, from performance and complexity point of views, a favorable situation corresponds to the availability of a large number of low-weight parity-checks.

In the following, x_n, $n = 1,2,...,N$, denotes an LFSR output sequence which is a codeword $\mathbf{x}$ of a binary (N,L) punctured simplex code $\mathbf{C}$ where N is codeword length and L is number of information bits. $\mathbf{x}_0 = [x_1, x_2, ..., x_L]$ is the vector of information bits identical to the LFSR initial state; $\{z_n\}$ denotes the degraded sequence $\{x_n\}$ after transmission over a BSC with crossover probability p. Accordingly, $z_n = x_n \oplus e_n \quad , \quad n = 1,2,...,N \quad$, where the effect of the BSC with error probability p is modeled by an N-dimensional binary random variable $\mathbf{E}$ defined over $\{0,1\}^N$ with independent coordinates E_n such that $\Pr(E_n = 1) = p$, $n = 1,2,\ldots,N$, and e_n is a realization of E_n. Applying a codeword $\mathbf{x} = [x_n]_{n=1}^N \in \mathbf{C}$, to the input of the BSC, we obtain the random variable $\mathbf{Z} = \mathbf{E} \oplus \mathbf{x}$ as a received codeword at its output. Let $\mathbf{z} = [z_n]_{n=1}^N$ and $\mathbf{e} = [e_n]_{n=1}^N$ denote particular values of the random vector variables $\mathbf{Z}$ and $\mathbf{E}$, respectively.

3 Novel Appropriate Parity-Check Sets

This section points out novel sets of the parity-check equations relevant for construction of an algorithm for the fast correlation attack which will be proposed in the next section. Also, this section points out the expected cardinalities of these sets.

3.1 Preliminaries

An LFSR can be considered as a linear finite state machine. Recall that a linear finite state machine is a realization or an implementation of certain linear operator. Accordingly, a state of a length-L LFSR after t clocks is given by the following matrix-vector product over GF(2):

$$\mathbf{x}_t = \mathbf{A}^t \mathbf{x}_0 \ , \quad t = 1, 2, ... \ ,$$

where $\mathbf{x}_t$ is an L dimensional binary vector representing the LFSR state after t clocks, $\mathbf{x}_0$ is an L dimensional binary vector representing the initial LFSR state (in notation that it has index L at the top and index 1 at the bottom), and $\mathbf{A}^t$ is the t-th power over GF(2) of the state transition $L \times L$ binary matrix $\mathbf{A}$. Assuming the LFSR characteristic polynomial $f(u) = 1 + \sum_{i=1}^{L} b_i u^i$, the matrix $\mathbf{A}$ is given by:

$$\mathbf{A} \ = \begin{bmatrix} b_1 & b_2 & b_3 & ... & & b_L \\ 1 & 0 & 0 & ... & & 0 \\ 0 & 1 & 0 & ... & & . \\ . & . & . & ... & & . \\ 0 & & & ... & 1 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \mathbf{A}_3 \\ . \\ \mathbf{A}_L \end{bmatrix} \ , \tag{1}$$

where each $\mathbf{A}_i$, $i = 1, 2, ..., L$, represents a $1 \times L$ binary matrix (a row-vector).

Powers of the matrix $\mathbf{A}$ determine algebraic replica of the LFSR initial state bits, i.e. linear equations satisfied by the bits of the codewords from the dual code. Accordingly, they directly specify the parity-checks.

Since our approach assumes an exhaustive search, over the first B information bits, the parity checks are obtained:
- directly from the powers of the matrix $\mathbf{A}$ corresponding to an arbitrary subset of the first B bits of the LFSR initial state and no more than three bits from the remaining $L - B$ bits of the initial state and the bit of the LFSR output sequence;
- as the *mod*2 sum of any two parity checks determined from the powers of the matrix $\mathbf{A}$ when this sum includes an arbitrary number of the first B bits of the LFSR initial state, at most one bit from the remaining $L - B$ bits of the initial state, and the two bits of the LFSR output sequence.
- as the *mod*2 sum of any three parity checks determined by the powers of matrix $\mathbf{A}$ when this sum includes an arbitrary number of the first B bits of the LFSR initial state, no bit from the remaining $L - B$ bits of the initial state, and the three corresponding bits of the LFSR output sequence.

As pointed out in Section 2, a desirable situation is that corresponding to as many low-weight parity-checks as possible. Following this fact and due to the comparison purposes with recently reported improved fast correlation attacks [8] - [9], we focus our intention mainly to parity-checks of effective weight three (i.e. without considering the first B bits), but also employ some of parity-checks of effective weight four as well. Note finally that parity-checks of an arbitrary weight could be considered as a development of certain reported results, for example [13] and [16].

3.2 Methods for Construction and Specification of the Parity-Check Sets

This section presents two methods for obtaining appropriate sets of parity-checks. The developed methods are related to the *information bits* (Method A) and to the *parity bits* (Method B) of the underlying punctured simplex code.

Method A: Parity-check sets related to the *information bits* of the underlying punctured simplex codeword.

Note that

$$x_{L+n} = \mathbf{A}_1^n \mathbf{x}_0 \ \ , \ \ n = 1, 2, ..., N - L \ \ , \tag{2}$$

where $\mathbf{A}_1^n$ is the first row of the n-th power of the state transition matrix $\mathbf{A}$.

Accordingly, the basic parity-check equations (defined on the noisy sequence) are given by:

$$c_{L+n} = z_{L+n} \oplus \mathbf{A}_1^n \mathbf{z}_0 \ \ , \ \ n = 1, 2, ..., N - L \ \ , \tag{3}$$

where $\mathbf{z}_0 = [z_1, z_2, ..., z_L]$.

Assuming that the first B information bits are known, appropriate parity-check equations for the i-th information bit, $i = B{+}1, B{+}2, ..., L$ can constructed according to the following definition.

Definition 1. The set Ω_i of parity-check equations associated with information bit-i is composed of:

- All parity-check equations corresponding to the vectors $\mathbf{A}_1^n$ such that each $\mathbf{A}_1^n$ has arbitrary values in the first B coordinates, has value one at the i-th coordinate, and has two ones in all other information bit coordinates;
- All parity-check equations obtained as the *mod*2 sum of two other basic parity-check equations,

  $$(z_m \oplus \mathbf{A}_1^m \mathbf{z}_0) \oplus (z_n \oplus \mathbf{A}_1^n \mathbf{z}_0) \ \ ,$$

  where m and n have arbitrary values providing that the vector sum $\mathbf{A}_1^m \oplus \mathbf{A}_1^n$ has arbitrary values in the first B coordinates, value one at the i-th coordinate, and value zero in the all other coordinates.

Note that for given parameters N, L, and B, the sets Ω_i, $i = B + 1, B + 2, ..., L$, can be constructed in advance through a search procedure in a preprocessing phase, and later used for any particular application with these given parameters.

Method B: Parity-check sets related to the *parity bits* of the underlying punctured simplex codeword.

First, an appropriate form of the parity check matrix of a punctured simplex code is pointed out. Then a method for constructing the parity checks is given and the parity checks to be employed by the algorithm are specified by Definition 2.

Recall, that in Section 2, the fast correlation attack has been modeled by the decoding of an (N, L) punctured simplex code used over a BSC. Accordingly, the following statement points out an appropriate form of the code parity-check matrix. This particular form has a one-to-one correspondence with the finite-state machine model of an LFSR with primitive characteristic polynomial.

Proposition 1. The parity-check matrix $\mathbf{H} = [\, \mathbf{P}^T, \mathbf{I}_{N-L} \,]$ of a punctured simplex code (N, L) with corresponding polynomial $f(u) = 1 + \sum_{i=1}^{L} b_i u^i$, where the binary matrix $\mathbf{P}$ is the $L \times (N-L)$ matrix of parity checks, $\mathbf{P}^T$ is its transpose, and $\mathbf{I}_{N-L}$ is the identity matrix of dimension $(N-L) \times (N-L)$, is specified by the following:

$$\mathbf{P}^T = \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \cdot \\ \cdot \\ \mathbf{P}_{N-L} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1^{(1)} \\ \mathbf{A}_1^{(2)} \\ \cdot \\ \cdot \\ \mathbf{A}_1^{(N-L)} \end{bmatrix} , \tag{4}$$

where the m-th row of the matrix $\mathbf{P}^T$, is an L-dimensional row vector $\mathbf{A}_1^{(m)}$ equal to the first row of the m-th power, $\mathbf{A}^m$, of the matrix $\mathbf{A}$ given in (1).

The construction of the parity-checks is based on searching for certain linear combinations of rows in an appropriate form of the parity-check matrix given by Proposition 1. Accordingly, the preprocessing phase of the algorithm includes the construction of the parity-checks according to the following algorithm which generates a set of parity checks for each parity bit. Each parity check includes certain B information bits, and no more than $W+1$ other arbitrary check bits.

Note that $W+1$ is used here instead three to illustrate that a straightforward generalization is possible where not only the parity-checks of effective weight equal to three are considered.

Algorithm for the construction of the parity checks

- *Input:* The parity check matrix $\mathbf{H} = [\, \mathbf{P}^T, \mathbf{I}_{N-L} \,]$.
- *Processing Steps:* For each parity bit, generate a set of parity check equations employing the following procedure.
  - For $n = L+1, L+2, ..., N$ and each w, $1 \leq w \leq W$, proceed as follows:
    - Calculate the *mod*2-sum of the n-th row of the parity-check matrix $\mathbf{H} = [\mathbf{P}^T, \mathbf{I}_{N-L}]$ and any possible w other rows.

  - If the values at positions $i = B+1, B+2, ..., L$, are all zeros, where $B < L$, is a predetermined parameter, record the considered combination into the set Ω_n^*.
- *Output:* The sets of parity check equations Ω_n^*, $n = L+1, L+2, ..., N$.

Definition 2: The set Ω_n^* generated by the above algorithm is the set of all considered parity-check equations related to the n-th parity bit of codewords in the punctured (N, L) simplex code.

Note that each parity-check in Ω_n^* consists of α of the first B information bits with $0 < \alpha \leq B$, none of the remaining last $L - B$ information bits and at most $W + 1$ of the $N - L$ parity check bits, including bit-n.

3.3 Expected Cardinalities of the Parity-Check Sets

Lemma 1. In any set Ω_i, specified by the Definition 1, $i = B+1, B+2, ..., L$, a tight approximation about the expected number $|\bar{\Omega}|$ of the parity-checks is given by the following:

$$|\bar{\Omega}| = 2^{B-L}[(N-L)\binom{L-B-1}{2} + \binom{N-L}{2}] \ . \tag{5}$$

Note that Lemma 1 motivates the construction of Ω_i given in Definition 1. For each type of check sums in Ω_i, that corresponding to minimum weight with non negligible contribution to $|\bar{\Omega}|$ is chosen.
As an illustration, note that for $N = 40000$, $L = 40$ and $B = 18, 19, 20, 21, 22$, Lemma 1 yields that the expected cardinality, $|\bar{\Omega}|$ is equal to 192, 384, 768, 1534, 3066, respectively.

Lemma 2: In any set Ω_n^*, specified by the Definition 2, $n = L+1, L+2, ..., N$, a tight approximation about the expected number $|\bar{\Omega}^*|$ of the parity-checks is given by the following:

$$|\bar{\Omega}^*| \ = \ 2^{-L+B} \sum_{w=1}^{2} \binom{N-L-1}{w} \ . \tag{6}$$

As an illustration, note that for $L = 40$, and (N, B) = (1024,26), (4096,22), (8192,20), and (16384,18), Lemma 2 yields that the expected cardinalities, $|\bar{\Omega}^*|$ are equal to 29.5, 31.4, 31.7, and 31.9, respectively.

Note that Lemmas 1 and 2 show that Definitions 1 and 2 yield large numbers of the parity-checks relevant for an error-correction procedure.

Also, note that Lemmas 1 and 2 imply that the expected cardinalities of the parity-check sets specified by Definitions 1 and 2 do not depend on the LFSR characteristic polynomial, and particularly on its weight.

4 Novel Algorithm for Fast Correlation Attack

The main underlying principles for construction of the novel fast correlation attack include the following:

- General concepts of linear block codes decoding, and particularly:
  - decoding of information bits only, employing an APP based threshold decoding;
  - iterative decoding of the parity bits employing a reduced complexity BP based iterative decoding.
- A novel method for constructing parity checks of a punctured simplex code based on linear finite state machine model of an LFSR (see [13]);
- The idea (implicitly given in [8]) of employing a partial (restricted) exhaustive search in order to enhance performance of the fast correlation attack. The developed algorithm assumes exhaustive search over the first B information bits in conjunction with appropriate decoding approaches.

According to these principles a novel algorithm for the fast correlation attack (based on a linear block code decoding approach) is proposed. The algorithm is based on the novel methods for constructing the appropriate parity-checks presented in the Section 3, and its processing phase includes the following three techniques: (i) hypothesis testing, (ii) decoding of a punctured simplex code and (iii) correlation check. The algorithm employs two different decoding procedures in order to provide desired trade-offs between necessary length of the sample, i.e. the rate of underlying code, performance and overall complexity.

Algorithm for the Fast Correlation Attack

INPUT:

- values of the parameters N, L, B, and the threshold T;
- the noisy received bits $z_1, z_2, ..., z_N$;
- for each information bit i, $i = B+1, B+2, ..., L$, the set Ω_i of corresponding parity-check equations (constructed in the preprocessing phase based on Definition 1), and for each parity bit n, $n = L+1, L+2, ..., N^*$, $N^* \leq N$, the set Ω_n of corresponding parity-check equations (constructed in the preprocessing phase based on Definition 2).

PROCESSING STEPS:

1. *setting the hypothesis*
   From the set of all possible 2^B binary patterns, select a not previously considered pattern $\hat{x}_1, \hat{x}_2, ..., \hat{x}_B$, for the first B information bits. If no new pattern is available, go to the Output (b).
2. *decoding*
   Employ one of the following two decoding algorithms for estimating a candidate for the information bits (i.e. LFSR initial state):

   - One-Step Decoding Algorithm (OSDA) using parity-checks specified by Definition 1;
   - Iterative Decoding Algorithm (IDA) using parity-checks specified by Definition 2.
3. *correlation check*
   Check if the current estimation of the information bits (obtained from the decoding step) $\hat{\mathbf{x}}_0 = [\hat{x}_1, \hat{x}_2, ..., \hat{x}_L]$, is the true one, according to the following:
   For $\hat{\mathbf{x}}_0$, generate the corresponding sequence $\hat{x}_1, \hat{x}_2, ..., \hat{x}_N$, and calculate $S = \sum_{n=1}^{N} \hat{x}_n \oplus z_n$.
   If $S \leq T$ go to Output (a), otherwise go to Step 1.

OUTPUT:
  (a) the considered vector $\hat{\mathbf{x}}_0$ of information bits is the true one;
  (b) the true vector of information bits is not found.

The threshold scalar T is used for checking a hypothesis over all the information bits. For given N, L, B, p, the threshold T is calculated based on the method presented in [18].

The specifications of the employed decoding algorithms OSDA and IDA are given in the following.

4.1 One-Step Decoding Algorithm - OSDA

OSDA decodes the noisy received sequence $[z_1, z_2, ..., z_N]$ for the (N, L) truncated simplex code employing an APP threshold decoding and the sets Ω_i of parity-check equations, specified by Definition 1, $i = B+1, B+2, ..., L$ according to the following.

- *parity-checks calculation*
  For each information bit position i, $i = B+1, B+2, ..., L$, calculate the parity-check values employing the parity check equations from the set Ω_i.
- *error-correction*
  For each i, $i = B+1, B+2, ..., L$ do the following:
  - if the number of satisfied parity-check equations for the considered information bit is smaller than the threshold $T_1(i)$ set $\hat{x}_i = z_i \oplus 1$, otherwise set $\hat{x}_i = z_i$.

The algorithm employs a vector threshold $\mathbf{T}_1 = [T_1(i)]_{i=B+1}^{L}$ which contains values for the APP threshold decoding of certain information bits.
Elements of the threshold vector $\mathbf{T}_1$ are determined based on the posterior error probabilities computed by using the parity-checks specified by Definition 1. We assume that for each codeword bit, the parity-checks used are orthogonal on that bit, meaning that except for that bit, every other involved unknown bit appears in exactly one of the parity-checks. Finally, assuming as an appropriate approximation, that all the parity-check equations involve exactly two unknown

bits beside the considered one, for any $i = B+1, B+2, ..., L$, the threshold $T_1(i)$ is equal to the smallest integer such that the following inequality holds:

$$\frac{p}{1-p}\left(\frac{1+(1-2p)^2}{1-(1-2p)^2}\right)^{|\Omega_i|-2T_1(i)} \leq 1 \ , \tag{7}$$

where $|\Omega_i|$ denotes the number of parity-check equations related to the i-th information bit [10].

4.2 Iterative Decoding Algorithm - IDA

For a given $N^* \leq N$, IDA decodes the received sequence $[z_1, z_2, ..., z_{N^*}]$ for the (N^*, L) punctured simplex code employing a BP based bit-flipping (BP-BF) iterative decoding and the sets Ω_n^* of parity-check equations, specified by Definition 2, $n = L+1, L+2, ..., N^*$.

BP-BF based iterative decoding (see [4], for example) includes the following main difference in comparison with simple BF.

- For each bit n, and each combination of $|\Omega_n^*| - 1$ parity-checks out of the $|\Omega_n^*|$ parity checks associated with bit-n, make $|\Omega_n^*|$ estimate of the nth bit value associated with these combinations.

Accordingly, we employ the following iterative BP-BF based decoding algorithm.

- *Initialization*: $\hat{x}_n = z_n$ and $\hat{x}_{nm} = z_n$.
- *Iterative Processing*
  1. *Step 1*:
     (a) For each n and for each $m \in \Omega_n^*$, evaluate:
     $\sigma_n(m) = \sum_{n' \in \omega(m)} \hat{x}_{n'm}$ $[mod2]$.
     (b) If all $\sigma_n(m) = 0$ go to Step 3 (a). If some maximum number of iterations (e.g. 30) is exceeded go to Step 3 (b).
  2. *Step 2*: For each n, do the following:
     (a) If $\sum_m^{|\Omega_n^*|} \sigma_n(m) \geq |\Omega_n^*|/2$, then $\hat{x}_n = \hat{x}_n \oplus 1$.
     (b) If $\sum_{m'}^{|\Omega_n^* \setminus m|} \sigma_n(m') \geq |\Omega_n^* \setminus m|/2$, then $\hat{x}_{nm} = \hat{x}_{nm} \oplus 1$.
     If no complementation was performed go to Step 3 (b); otherwise go to Step 1.
  3. *Step 3*:
     (a) $\hat{\mathbf{x}} = [\hat{x}_n]$ is the decoding result.
     (b) Algorithm halts and a warning is declared that a valid decoding is not reached.

5 Performance and Complexity

5.1 Performance

The performance of the novel algorithm is experimentally considered when the LFSR characteristic polynomial is chosen as $1+u+u^3+u^5+u^9+u^{11}+u^{12}+u^{17}+$

Table 1. Performance of the novel algorithm - experimental analysis: Error-rate of the LFSR initial state reconstruction, as a function of the correlation noise p when the LFSR length is $L = 40$, the characteristic polynomial weight is 17, and the length of the sequence available for processing is $N = 40000$ bits.

p	Error rate of LFSR initial state reconstruction					
	OSDA $B = 18$	OSDA $B = 19$	OSDA $B = 20$	OSDA $B = 21$	OSDA $B = 22$	IDA $N^* = 4096$, $B = 22$
0.25	0.009	0.000	0.000	0.000	0.000	0.000
0.26	0.015	0.000	0.000	0.000	0.000	0.000
0.27	0.024	0.000	0.000	0.000	0.000	0.000
0.28	0.081	0.000	0.000	0.000	0.000	0.000
0.29	0.159	0.006	0.000	0.000	0.000	0.000
0.30	0.254	0.023	0.000	0.000	0.000	0.000
0.31	0.384	0.041	0.002	0.000	0.000	0.000
0.32	0.569	0.098	0.002	0.000	0.000	0.000
0.33	0.696	0.226	0.020	0.000	0.000	0.000
0.34	0.838	0.356	0.053	0.001	0.000	0.000
0.35	0.915	0.542	0.114	0.002	0.001	0.000
0.36	0.955	0.743	0.225	0.019	0.022	0.000
0.37	0.983	0.865	0.450	0.080	0.062	0.001
0.38	0.990	0.932	0.652	0.210	0.208	0.023
0.39	0.997	0.980	0.850	0.445	0.399	0.052
0.40	1.000	0.988	0.935	0.663	0.651	0.267

$u^{19} + u^{21} + u^{25} + u^{27} + u^{29} + u^{32} + u^{33} + u^{38} + u^{40}$ and $N = 40000$ (i.e. assuming the same example as was considered in [8]-[9]). Note that the proposed algorithm can be applied for values of L significantly longer than $L = 40$, but this value was employed in all numerical and experimental illustrations for comparison with previously reported results.

Results of the performance analysis are presented in Table 1. This table displays the error-rate of the LFSR initial state reconstruction as a function of the correlation noise p when the algorithm employs:
(i) OSDA with B = 18, 19, 20, 21, 22,
(ii) IDA with $N^* = 4096$, $B = 22$, and at most 20 iterations.

Each error-rate given in the table is obtained by calculation over a corresponding, randomly selected, set of 1000 samples. Recall that "error-rate" indicates the fraction of trials for which we obtain incorrect decoding (and accordingly incorrect reconstruction of the secret key).

5.2 Complexity

Recall that the overall complexity assumes time and space complexity requirements. The complexity analysis yields that according to the structure of the proposed algorithm:
- The algorithm requires a space for the input. Space requirements for the decoding process are as follows: when OSDA is employed, decoding processing does not require memory; IDA requires a memory proportional to the parameter N^*;
- Time complexity is specified by the following corollaries.

Corollary 1. Assuming that OSDA is employed, that $|\Omega|$ denotes the average cardinality of the parity-check sets $|\Omega_i|$, that ω denotes the average number of bits in a parity-check, and that w denotes the weight of the LFSR characteristic polynomial, the implementation complexity of the proposed algorithm is proportional to $2^B[\ (L-B)|\Omega|\omega\ +\ (N-L)w\ ]\ mod2$ additions.

Corollary 2. Assuming that IDA is employed, that $|\Omega^*|$ denotes the average cardinality of the parity-check sets $|\Omega_n|$, that ω^* denotes the average number of bits in a parity-check, that I denotes the number of iterations, and that w denotes the weight of the LFSR characteristic polynomial, the implementation complexity of the proposed algorithm is proportional to $2^B[\ I(N^*-L)|\Omega^*|(|\Omega^*|-1)\omega^*\ +\ (N-L)w\ ]\ mod2$ additions.

Note also that from the structure of the proposed algorithms, it is readily seen that the proposed algorithms are suitable for fast software implementation, as well as for simple hardware implementation: the algorithms employ only simple arithmetic operations (*mod*2 addition) and simple logical operations.

Also, since the decoding process is mainly memoryless, note that a reduction of the time complexity specified by the previous corollaries can be obtained by an appropriate time-memory complexity trade-off.

Finally note that in the presented experiments, the decoding step has employed the underlying codeword lengths $N = 40000$ and $N^* = 4096$ for OSDA and IDA, respectively. This is an illustration that OSDA and IDA yield a trade-off between the length of the required sample (i.e. the code rate) and the decoding complexity.

6 Comparison of the Novel Algorithm with Recently Proposed Improved Fast Correlation Attacks

This section presents an arguably comparative analysis of the underlying principles, performance and complexity of recently proposed improved fast correlation attacks [9] and the novel algorithm, assuming the same input.

6.1 Comparison of the Underlying Principles

Comparison of the underlying principles employed in [8]-[9] and in the novel algorithm for the fast correlation attack can be summarized as follows.

- The approaches of [8]-[9] are based on decoding of convolutional codes and turbo codes with convolutional codes as the component codes constructed over the LFSR sequence. The novel approach is based on decoding punctured simplex block codes corresponding to the LFSR sequence.
- The algorithms [8]-[9] and the novel algorithm employ different parity-checks. The parity-checks employed in [9]-[8] are constructed by searching for these parity checks which include the following bits: currently considered bit, bits from a subset of B previous bits, and no more than two other bits.
  The parity-checks employed in the novel algorithm are constructed by searching for these parity checks which include the following bits:
  (i) currently considered information bit, bits from a subset of B first information bits, and two other information bits with the corresponding parity-bit, or two arbitrary parity bits only, or
  (ii) currently considered parity bit, bits from a subset of B first information bits, and no more than two other parity bits.
  Note that these different approaches in the parity-check constructions imply different number of parity-checks per bit, as well.
- The decoding techniques employed in [8]-[9] are Viterbi decoding, BCJR decodings, and MAP turbo decoding (see [7] and [1]). On the other hand the novel algorithm employs the following two low-complexity decoding techniques: (i) APP threshold decoding, and (ii) BP based BF iterative decoding.
- The fast correlation attacks from [8]-[9] implicitly include an exhaustive search over a set of dimension 2^B through employment of the Viterbi or BCJR decodings due to the trellis search. The novel algorithm employs an explicit search over all 2^B possible patterns corresponding to the first B information bits.
- A decoding process based on the Viterbi or BCJR algorithm requires a memory of dimension proportional to 2^B. On the other hand, OSDA does not require memory, and IDA requires a memory proportional to the parameter N^*.

6.2 Comparison of the Performance and Complexity

For the performance comparison of the novel and turbo based fast correlation attacks [9] the same inputs are employed and relevant parameters are selected so that the novel algorithm always has significantly lower overall implementation complexity than the algorithm [9].

According to [9], the time complexity of the turbo decoding is proportional to $2^B IMJm$ real multiplications where I denotes the number of the iterations,

Table 2. Comparison of the algorithms performance, assuming the same inputs, and lower complexity of the novel algorithm in comparison to the turbo algorithm [9]: Limit noise for which the algorithms yield, with probability close to 1, correct reconstruction of the initial LFSR state, when the LFSR characteristic polynomial is $1 + u + u^3 + u^5 + u^9 + u^{11} + u^{12} + u^{17} + u^{19} + u^{21} + u^{25} + u^{27} + u^{29} + u^{32} + u^{33} + u^{38} + u^{40}$, and the available sample is 40000 bits.

ALGORITHM	Limit Noise
turbo algorithm [9]: $B = 15$, $M = 2$	0.27
novel algorithm with OSDA: $B = 19$	0.28
turbo algorithm [9]: $B = 15$, $M = 4$	0.29
novel algorithm with OSDA: $B = 21$	0.33
turbo algorithm [9]: $B = 15$, $M = 16$	0.30
novel algorithm with OSDA: $B = 22$	0.34
novel algorithm with IDA: $N^* = 4096$, $B = 22$	0.36

M the number of the component codes, J the number of processed bits, and m the number of employed parity-checks per bit. The time complexity of the novel algorithm is given in Corollaries 1 and 2.

Also note that the space complexity of the approach from [9] is proportional to 2^B due to employment of the BCJR algorithm. If OSDA is employed no space complexity is required, and if IDA is employed it is usually significantly smaller than 2^B due to its linear rather than exponential nature.

An illustrative performance comparison is presented in the Table 2. Note that, in each case, the complexity of the proposed algorithm could be considered as significantly lower than complexity of the turbo decoding [9] although the proposed algorithm assumes search over a much larger set of hypotheses, since: (i) [9] employs iterative processing with M component codes and (ii) the dominant arithmetic operation in the proposed algorithm is $mod2$ addition against real multiplication for the turbo based decoding of [9].

Finally, note that the actual time for performing the attack by the novel algorithm strongly depends on the implementation constraints so that a straightforward comparison is not appropriate. Also, the approaches of [8]-[9] can be modified to involve $mod2$ additions, but at the expense of performance degradation.

7 Conclusions

A novel algorithm for the fast correlation attack has been proposed. The algorithm is based on decoding procedures of the corresponding binary block code with novel constructions of the parity-checks, independent of the LFSR feedback polynomial weight, and the following two decoding approaches are employed: an APP based threshold decoding and a BP based BF iterative decoding . The constructions of the parity-checks are based on searching for certain parity-check equations and their linear combinations employing the finite-state machine model of an LFSR with primitive characteristic polynomial. The expected numbers of the parity-checks per parity bit have been derived, showing that a large number of appropriate parity-checks can be constructed.

The performance of the proposed algorithm has been analyzed experimentally showing that the algorithm is a powerful one.

The overall implementation complexity has been specified. As dominant operations the algorithm employs *mod*2 additions and simple logical operations, so that it is very suitable for high-speed software implementation as well as for simple hardware implementation.

The algorithm offers good trade-offs between required sample length (i.e. rate of the underlying code), overall complexity and performance. The one-step threshold decoding approach yields high performance assuming long enough sample, and the iterative decoding approach can reach the same performance using a significantly shorter sample but at the expense of increased complexity.

The novel algorithm has been compared with recently reported improved fast correlation attacks based on convolutional codes and turbo decoding. The underlying principles, performance and complexity have been compared, and the essential gain obtained with the novel approach is pointed out. The developed algorithm has the following two main advantages over other reported ones:
(a) Assuming a lower overall complexity, and the same inputs, the proposed algorithm yields significantly better performance.
(b) It is suitable for high-speed software implementation as well as for simple hardware implementation and highly parallel architectures.

References

1. L. Bahl, J. Cocke, F. Jelinek and J. Raviv, “Optimal decoding of linear codes for minimizing symbol error rate,” *IEEE Transactions on Information Theory*, vol. IT-20, pp. 284-287, March 1974.
2. V. Chepyzhov and B. Smeets, ”On fast correlation attack on certain stream ciphers”, Advances in Cryptology - EUROCRYPT ’91, *Lecture Notes in Computer Science*, vol. 547, pp. 176-185, 1991.
3. A. Clark, J. Dj. Golić, and E. Dawson, ”A comparison of fast correlation attacks,” Fast Software Encryption - FSE’96, *Lecture Notes in Computer Science*, vol. 1039, pp. 145-157, 1996.
4. R. G. Gallager, ”Low-density parity-check codes,” *IRE Transactions on Information Theory*, vol. IT-8, pp. 21-28, Jan. 1962.
5. M. P. C. Fossorier, M. J. Mihaljević and H. Imai, ”Reduced complexity iterative decoding of Low Density Parity Check codes based on Belief Propagation”, *IEEE Transactions on Communications*, vol. 47, pp. 673-680, 1999.

6. M. P. C. Fossorier, M. J. Mihaljević and H. Imai, "Critical noise for convergence of iterative probabilistic decoding with belief propagation in cryptographic applications", Applied Algebra, Algebraic Algorithms and Error Correcting Codes - AAECC 13, *Lecture Notes in Computer Science*, vol. 1719, pp. 282-293, 1999.
7. R. Johannesson and K. Zigangirov, *Fundamentals of Convolutional Coding.* New York: IEEE Press, 1999.
8. T. Johansson and F. Jonsson, "Improved fast correlation attacks on stream ciphers via convolutional codes", Advances in Cryptology - EUROCRYPT'99, *Lecture Notes in Computer Science*, vol. 1592, pp. 347-362, 1999.
9. T. Johansson and F. Jonsson, "Fast correlation attacks based on turbo code techniques", Advances in Cryptology - CRYPTO'99, *Lecture Notes in Computer Science*, vol. 1666, pp. 181-197, 1999.
10. J. L. Massey, *Threshold Decoding.* Cambridge, MA: MIT Press, 1963.
11. A. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography.* Boca Raton: CRC Press, 1997.
12. W. Meier and O. Staffelbach, "Fast correlation attacks on certain stream ciphers," *Journal of Cryptology*, vol. 1, pp. 159-176, 1989.
13. M. J. Mihaljević and J. Dj. Golić, "A fast iterative algorithm for a shift register initial state reconstruction given the noisy output sequence", Advances in Cryptology - AUSCRYPT '90, *Lecture Notes in Computer Science*, vol. 453, pp. 165-175, 1990.
14. M. J. Mihaljević and J. Dj. Golić, "A comparison of cryptanalytic principles based on iterative error-correction," Advances in Cryptology - EUROCRYPT '91, *Lecture Notes in Computer Science*, vol. 547, pp. 527-531, 1991.
15. M. J. Mihaljević, M.P.C. Fossorier and H. Imai, "Novel fast correlation attack via iterative decoding of punctured simplex code", Proceedings of *IEEE ISIT'2000*, Sorento, Italy, June 2000.
16. M. J. Mihaljević and J. Golić, "A method for convergence analysis of iterative probabilistic decoding", accepted for publication in *IEEE Transactions on Information Theory.*
17. W. Penzhorn, "Correlation attacks on stream ciphers: Computing low-weight parity checks based on error-correcting codes", Fast Software Encryption - FSE'96, *Lecture Notes in Computer Science*, vol. 1039, pp. 159-172, 1996.
18. T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only", *IEEE Transactions on Computers*, vol. C-34, pp. 81-85, 1985.
19. K. Zeng and M. Huang, "On the linear syndrome method in cryptanalysis," Advances in Cryptology - CRYPTO '88, *Lecture Notes in Computer Science*, vol. 403, pp. 469-478, 1990.
20. K. Zeng, C.H. Yang and T.R.N. Rao, "An improved linear syndrome method in cryptanalysis with applications," Advances in Cryptology - CRYPTO '90, *Lecture Notes in Computer Science*, vol. 537, pp. 34-47, 1991.

Improved Cryptanalysis of Rijndael

Niels Ferguson[1], John Kelsey[1], Stefan Lucks[⋆2], Bruce Schneier[1], Mike Stay[3], David Wagner[4], and Doug Whiting[5]

[1] Counterpane Internet Security, Inc., 3031 Tisch Way Suite 100PE, San Jose, CA 95128
[2] University of Mannheim, 68131 Mannheim, Germany
[3] AccessData Corp. 2500 N. University Ave. Ste. 200, Provo, UT 84606
[4] University of California Berkeley, Soda Hall, Berkeley, CA 94720
[5] Hi/fn, Inc., 5973 Avenida Encinas Suite 110, Carlsbad, CA 92008

Abstract. We improve the best attack on Rijndael reduced to 6 rounds from complexity 2^{72} to 2^{44}. We also present the first known attacks on 7- and 8-round Rijndael. The attacks on 8-round Rijndael work for 192-bit and 256-bit keys. Finally, we discuss the key schedule of Rijndael and describe a related-key attack that can break 9-round Rijndael with 256-bit keys.

1 Introduction

Rijndael is one of the five AES candidate ciphers that made it to the second round [DR98]. Rijndael has 10, 12, or 14 rounds, depending on the key size. Previously it was known how to break up to 6 rounds of Rijndael [DR98]. Independently from our work, Gilbert and Minier [GM00] presented an attack on 7 rounds of Rijndael.

In section 2, we describe a new *partial sum* technique that can dramatically reduce the complexity of the 6-round attacks. We also show how to use these ideas to attack 7 and 8 rounds of Rijndael, in some cases using additional known texts (where available) to reduce the workfactor. The attacks against 7-round Rijndael with 128-bit keys and 8-round Rijndael with 192-bit and 256-bit keys require nearly the entire Rijndael codebook ($2^{128} - 2^{119}$ chosen plaintexts); they are therefore not very practical even for an adversary with sufficient computing power. All of these attacks use extensions of the dedicated Square attack, as described in [DKR97,DR98,DBRP99].

In section 3, we turn our attention to the key schedule. We show several unexpected properties of the key schedule that seem to violate the published design criteria. Although we do not know of any attacks that critically depend on these properties, we consider them unsettling. Finally, in section 4, we exploit the slow diffusion of the Rijndael key schedule to develop a related-key attack that can be mounted on 9 rounds of Rijndael with a 256-bit key.

A summary of these attacks, including time and data complexities, is described in table 1. We also refer the reader to appendix A for a detailed listing of notation used to refer to intermediate values in the cipher.

⋆ Supported by DFG grant KR 1521/3-2.

Table 1. Summary of Attacks on Rijndael.

Cipher	Key size	Complexity [Data]	[Time]	Comments
Rijndael-6	(all)	2^{32} CP	2^{72}	[DR98] (previously known)
Rijndael-6	(all)	$6 \cdot 2^{32}$ CP	2^{44}	partial sums (new)
Rijndael-7	(192)	$19 \cdot 2^{32}$ CP	2^{155}	partial sums (new)
Rijndael-7	(256)	$21 \cdot 2^{32}$ CP	2^{172}	partial sums (new)
Rijndael-7	(all)	$2^{128} - 2^{119}$ CP	2^{120}	partial sums (new)
Rijndael-8	(192)	$2^{128} - 2^{119}$ CP	2^{188}	partial sums (new)
Rijndael-8	(256)	$2^{128} - 2^{119}$ CP	2^{204}	partial sums (new)
Rijndael-9	(256)	2^{85} RK-CP	2^{224}	related-key attack (new)

CP – chosen plaintext, RK-CP – related-key chosen plaintext.

2 The Square Attack

2.1 The Original 6-Round Attack

We start by describing the 6-Round attack in the original proposal [DR98], which uses a technique first introduced to attack the block cipher Square [DKR97]. This is an attack that works against all block sizes and key sizes.

We use $m^{(r)}$, $b^{(r)}$, and $t^{(r)}$ to refer to intermediate text values used in round r after the MixColumn, key addition, and ShiftRow operations, respectively. We write $k^{(r)}$ for the subkey in round r, and $k^{(r)\prime}$ for an equivalent subkey value that may be XORed into the state before instead of after the MixColumn operation in round r. Please refer to appendix A for a more detailed explanation of our notation.

The attack starts by obtaining 256 encryptions that only differ in a single byte of $m^{(1)}$, and that take on all values for that particular byte. One byte of $m^{(1)}$ depends on four bytes of the plaintext and four bytes of $k^{(0)}$. We first choose 2^{32} plaintexts by taking a fixed starting point and varying those four bytes over all 2^{32} possible values. We then guess the four key bytes that are involved. For each possible value of the key bytes, we can find 2^{24} groups of 256 plaintexts such that within each group the encryptions differ in a specific byte of $m^{(1)}$; as the plaintexts are different, this one byte of $m^{(1)}$ must take on all 256 possible values.

Tracking these changes through the cipher, we find that each of the bytes of $t^{(4)}$ takes on all possible values. For each of these bytes, if we sum the values it takes on in the 256 encryptions, we get zero. This property is preserved by a linear function, so each of the bytes of $m^{(4)}$, and of $b^{(4)}$, also sums to zero over our 256 encryptions.

We now look at a particular byte of $b^{(4)}$, and how that relates to the ciphertext. For our analysis we rewrite the cipher slightly, and put the AddRoundKey before the MixColumn in round 5. Instead of applying MixColumn and then adding in $k^{(5)}$, we first add in $k^{(5)'}$ and then apply MixColumn. In this configuration it is easy to see that any byte of $b^{(4)}$ depends on the ciphertext, four bytes from $k^{(6)}$, and one byte from $k^{(5)'}$. We guess these five key bytes, compute the value of our $b^{(4)}$ byte for our 256 encryptions and check whether the sum is zero.

For each group of 256 plaintexts, this filter rejects 255/256 of all wrong key guesses. As we guess a total of nine key bytes, we will need 10 or so groups of 256 encryptions to find the key. (Note that these groups depend on the first four key bytes that we guessed, but not on the last five.)

Overall, this attack requires 2^{32} chosen plaintexts, 2^{32} memory to store those plaintext/ciphertext pairs, and 2^{72} steps in guessing the nine key bytes. Each step involves a partial decryption of 256 ciphertexts, but a proper ordering of these computations can make that very fast. This seems to be comparable to doing a single encryption, and agrees with the complexity estimate given in [DR98]. The overall complexity is thus comparable to 2^{72} encryptions.

2.2 A 7-Round Extension

This attack can be extended to 7 rounds for 192- and 256-bit keys. One simply guesses the 16 bytes of the last round key. When used naively, this adds 128 bits to the key guessing, for a total workload of 2^{200}; the plaintext and memory requirements are not changed, although we do need to use more groups to verify the potential keys.[1]

This can be further improved. The key schedule ensures that there are dependencies between the expanded key bytes, and we can exploit them in this attack. For a 192-bit key, guessing the last round key $k^{(7)}$ gives us two of the four bytes from $k^{(6)'}$ that we would otherwise have to guess plus the byte from $k^{(5)'}$ that we would guess. This saves us 24 bits of key guessing, and results in an overall complexity of 2^{176}. For 256-bit keys, the bytes in the key schedule are aligned differently. Guessing all of $k^{(7)}$ provides no information about $k^{(6)'}$ but does give us the one byte from $k^{(5)'}$ that we need. For this key length, the complexity of the attack thus becomes 2^{192}. All the details can be found in [Luc00].[2]

2.3 An Improvement

The attack of section 2.1 on 6 rounds of Rijndael can be improved. Instead of guessing four bytes of $k^{(0)}$ we simply use all 2^{32} plaintexts. For any value of the first round key, these encryptions consist of 2^{24} groups of 2^8 encryptions that vary only in a single byte of $m^{(1)}$. All we have to do is to guess the five key bytes

[1] For this attack we use the alternate round representation for both rounds 5 and 6, and thus add $k^{(6)'}$ before the MixColumn in round 6.

[2] Note that the attack complexities in [Luc00] are given in S-box lookups, whereas we roughly approximate the complexity of a single encryption by 2^8 S-box lookups and use encryptions as our unit. The result is that all our complexity numbers are a factor of 2^8 lower.

at the end of the cipher, do a partial decrypt to a single byte of $b^{(4)}$, sum this value over all the 2^{32} encryptions, and check for a zero result. Compared to the original version, we guess only 40 bits of key instead of 72. On the other hand, we have to do 2^{24} times as much work for each guess. All in all, this improvement reduces the workload by a factor of 2^8, although it needs about $6 \cdot 2^{32}$ plaintexts to provide enough sets of 2^{32} plaintexts to uniquely identify the proper value for the five key bytes.

We will now look at this attack in more detail. We have 2^{32} ciphertexts. We guess five key bytes, do a partial decryption from each of the ciphertexts to a single byte in $b^{(4)}$, and sum this byte over all ciphertexts. Consider this partial decryption. From any ciphertext, we use four ciphertext bytes. Each of these is XORed with a key byte. We then apply the inverse S-box to each byte, and multiply each with an appropriate factor from the inverse MDS matrix. The four bytes are then XORed together, a fifth key byte is XORed into the result, the inverse S-box is applied, and the resulting value is summed over all ciphertexts.

Let $c_{i,j}$ be the jth byte of the ith ciphertext. (We leave out the i subscript if we are not talking about any particular ciphertext.) For simplicity we will number the four bytes of each ciphertext that we use from 0 to 3. Let $k_0, \ldots, k_4$ denote the five key bytes that we are guessing. We want to compute

$$\sum_i S^{-1}[S_0[c_{i,0} \oplus k_0] \oplus S_1[c_{i,1} \oplus k_1] \oplus S_2[c_{i,2} \oplus k_2] \oplus S_3[c_{i,3} \oplus k_3] \oplus k_4] \qquad (1)$$

where $S_0, \ldots, S_3$ are bijective S-boxes, each of which consists of an inverse Rijndael S-box followed by a multiplication by a field element from the inverse MDS matrix. Given 2^{32} ciphertexts and 2^{40} possible key guesses, we have to sum 2^{72} different values, which corresponds roughly in amount of work to doing about 2^{64} trial encryptions.

We can organize this more efficiently in the following manner. For each k, we associate a "partial sum" x_k to each ciphertext c, defined as follows:

$$x_k := \sum_{j=0}^{k} S_j[c_j \oplus k_j]$$

This gives us a map $(c_0, c_1, c_2, c_3) \mapsto (x_k, c_{k+1}, \ldots, c_3)$ that we can apply to each ciphertext if we know $k_0, \ldots, k_k$.

We start out with a list of 2^{32} ciphertexts. We guess k_0 and k_1 and compute how often each triple (x_1, c_2, c_3) occurs in the list. That is, for each i, we compute the three-byte value $(S_0[c_{i,0} \oplus k_0] \oplus S_1[c_{i,1} \oplus k_1], c_{i,2}, c_{i,3})$ as a function of the ith ciphertext and the guessed key material, and we count how many times each three-byte value appears during this computation. As there are only 2^{24} possible values for three bytes, we do not have to list all (x_1, c_2, c_3) values; rather, we count how often each triple occurs. We then guess k_2 and compute how often each tuple (x_2, c_3) occurs; and guess k_3 and compute how often each value of x_3 occurs. Finally, we guess k_4 and compute the desired sum.

Because all sums are taken using the XOR operation, and because $z \oplus z = 0$ for all z, it suffices to only count modulo two. Thus, a single bit suffices for each count, and so the space requirement for the 2^{24} counters is just 2^{24} bits.

How much work has this been? In the first phase we guessed 16 bits and processed 2^{32} ciphertexts, so this phase costs 2^{48} overall. In the next phase, we guessed a total of 24 bits but we only had to process 2^{24} triples, so this costs 2^{48} as well. This holds similarly for each of the phases. In total, the entire computation requires the equivalent of about 2^{48} evaluations of equation 1, or about 2^{50} S-box applications.

This is the amount of work required for a single structure of 2^{32} ciphertexts. The first structure already weeds out the overwhelming majority of the wrong key guesses, but we still have to do the first steps of our partial sum computation for each of the six structures that we use. The total number of S-box lookups is thus about 2^{52}.

Using our earlier rough equivalence of 2^8 S-box applications to a trial encryption with a new key, the 2^{52} S-box applications are comparable to 2^{44} trial encryptions. This is a significant improvement over the earlier 2^{72} workfactor.

2.4 Extension to 7 Rounds

We can apply this our improvement to the 7-round attack of section 2.2. To express a single byte of $b^{(4)}$ in the key and the ciphertext, we get a formula similar to equation 1 but with three levels, 16 ciphertext bytes, and 21 key bytes. The partial sum technique is only helpful during the last part of the computation as it only saves work if there are more ciphertexts than possible values for the intermediate result. With 2^{32} plaintext/ciphertext pairs in a structure, these techniques will not help until the very last part of the computation.

For 192-bit keys we first guess the 128 bits of the last round key. These guesses also define two of the four key bytes in round 6 that we are interested in, and the one key byte in round 5 that we need. Thus, after guessing the last round key we can reduce each structure to 2^{24} counters with our partial sum technique. Using some precomputed tables, we can do this for each of the 2^{128} key guesses in about 2^{32} memory lookups. The next phase guesses one byte more and requires 2^{24} steps to reduce the partial sum to 2^{16} counters, and the last phase guesses the last remaining byte and produces the final result. Each of these phases has a cost of 2^{160} lookups. We have three phases, each of which costs 2^{160}, and we need to process three structures before we start eliminating guesses for the last round key, so the overall cost of this attack is on the order of 2^{163} S-box lookups or about 2^{155} trial encryptions.

For 256-bit keys the alignment in the key schedule is different. Guessing the last round key does not give us any information about the round key of round 6, but it provides most of the round key for round 5. Working in a similar fashion as before, we guess 128 bits of the last round key and compute the four bytes we are interested in after round 6 for each of the 2^{32} texts for a total cost of 2^{160} lookups. The next phase guesses 16 more key bits and results in 2^{24} one-bit counters for a total cost of 2^{176} lookups. The remaining phases have a similar cost. The cost per structure is thus about 2^{178} lookups or about 2^{170} trial encryptions. We need five structures before we start cutting into the guesses of the last round key, so the overall complexity of this attack is about 2^{172}.

2.5 A Second Improvement

It is possible to push these attacks even further, if we are willing to trade texts for time and increase the data complexity to save on the workfactor.

We first show that 7 rounds of Rijndael may be broken with 2^{128} known texts (the entire codebook!) and workfactor equivalent to approximately 2^{120} trial encryptions. These encryptions consist of 2^{96} packs of 2^{32} encryptions that vary only in four bytes of $m^{(1)}$. Those four bytes are in a proper position to apply the attack of section 2.3: specifically, each pack of 2^{32} encryptions consists of 2^{24} groups of 2^8 encryptions that vary only in a single byte of $m^{(2)}$. Equivalently, we may view the entire set of 2^{128} encryptions as consisting of 2^{120} groups of 2^8 encryptions that vary only in one byte of $m^{(2)}$. This ensures that summing a single byte in $b^{(5)}$ over the 2^8 encryptions in a group yields zero, and thus summing over all 2^{128} encryptions also yields zero in this byte. This simple property is the basis for several attacks, as described below.

A naive way that one might try to exploit this property is to guess five key bytes at the end of the cipher, partially decrypt each ciphertext to a single byte of $b^{(5)}$, sum over all 2^{128} ciphertexts, and check for zero. However, the naive approach does not actually work. Even the wrong keys will yield zero when summing the byte in $b^{(5)}$ over all 2^{128} encryptions, because for any bijective 128-bit block cipher, $b^{(5)}$ (or any other intermediate value) will take on all possible 128-bit values as you cycle through all 2^{128} encryptions. Consequently, we will need to modify the attack slightly.

Instead, we use the following technique. Focus our attention on a fifth byte in $m^{(1)}$ (different from the four bytes selected earlier), say, $m^{(1)}_{a,b}$. Fixing a value x for this byte gives us a set of 2^{120} encryptions where $m^{(1)}_{a,b} = x$; this gives us a list of 2^{88} packs, where each pack contains 2^{24} groups of 2^8 encryptions that vary only in a single byte of $m^{(2)}$. We call this structure of 2^{120} encryptions a *herd*. Now we obtain 2^{128} known texts (2^8 herds), guess four key bytes at the beginning of the cipher, calculate $m^{(1)}_{a,b}$ for each encryption using our guessed key material, and separate the texts into herds. Examining a single such herd, we find that summing a byte in $b^{(5)}$ over all the encryptions in the herd yields zero, and moreover this property is unlikely to hold if our guesses at the key were incorrect. This yields a working attack against 7 rounds of Rijndael, but the complexity is very high ($2^{128} \times 2^{72}$ steps of computation or so).

One can do much better. Note that the byte in $b^{(5)}$ depends only on four bytes of the ciphertext (for simplicity, call them $c_0, \ldots, c_3$) and the byte $m^{(1)}_{a,b}$ depends on only four bytes of the plaintext ($p_4, \ldots, p_7$, say). We use a three-phase attack; the first phase uses 2^{64} counters (the m_y's), the second phase uses 2^{32} counters (the n_z's), and the third phase provides the filtering information for key guesses. As usual, all counters may be taken modulo 2, so we need just one bit for each counter.

The attack goes as follows. In the first phase, we increment the counter m_y corresponding to the 64-bit quantity $y = (c_0, \dots, c_3, p_4, \dots, p_7)$ as we see each known text (p, c). The second phase guesses four key bytes from the first round, separates the counters into herds (by computing $m_{a,b}^{(1)}$ for each counter position using $(p_4, \dots, p_7)$ and the guessed key material), selects a single herd, and updates the counter n_z by adding m_y to it for each y that is in the correct herd and that agrees with $z = (c_0, \dots, c_3)$. Afterwards, in the third phase, we guess five key bytes at the end of the cipher, partially decrypt each z to a single byte in $b^{(5)}$, sum this byte over all 2^{32} values of z (with multiplicities as given by the n_z), and check for zero. The third phase must be repeated for each guess of the four key bytes in the first round.

What is the complexity of this attack? The first phase requires us to update a counter for each ciphertext, so using our rough equivalence of 2^8 memory lookups to a trial encryption, the counting should take time comparable to 2^{120} trial encryptions. Compared to the first phase, the rest of the attack has negligible workfactor (equivalent to 2^{96} encryptions); there is no need to compute partial sums, an exhaustive key search will suffice.

This shows that one may break 7 rounds of Rijndael using 2^{128} known texts, 2^{120} work, and 2^{64} bits of memory. This 7-round attack trades texts for time: it uses a huge number of known texts, but it has better workfactor and overall complexity than the 7-round attack of section 2.3; and moreover, it applies to all key sizes (including the 128-bit keys).

There are a few more small improvements. We used a single byte of $m^{(1)}$ to define our herds, but the four plaintext bytes that we use in our attack and the four key bytes of the first round key that we guess define four bytes of $m^{(1)}$. We can create more (but smaller) herds by fixing three bytes of $m^{(1)}$ for each herd. This gives us 2^{24} herds of 2^{104} texts each.[3] We can even choose which of the four bytes will take on every value, and thus create 2^{26} herds of 2^{104} texts each, in which case each text is used in four different herds. Furthermore, we do not need all the plaintext/ciphertext pairs. If the four plaintext bytes take on $2^{32} - 2^{23}$ of the 2^{32} possible values (and for each of these values the other 12 bytes take on all possible values), then about half of our herds will have missing plaintext/ciphertext pairs while the other half are complete and undamaged. We can use the undamaged herds in our attack. This reduces the plaintext requirements to $2^{128} - 2^{119}$ texts. These changes do not change the complexity of the attack, but give us a slight reduction in the text requirements.

2.6 Extension to 8 Rounds

We can further extend the idea to break 8 rounds of Rijndael, though apparently not for 128-bit keys. As before, we obtain $2^{128} - 2^{119}$ texts (about 2^{23} undamaged

[3] Note that we cannot use all four bytes, as at least one of the four bytes has to vary within the pack.

herds), focus attention on a single herd, and use the fact that a single byte in $b^{(5)}$ will yield zero when summed over all 2^{104} encryptions in the herd. However, the byte in $b^{(5)}$ now depends on the entire ciphertext and on 21 subkey bytes at the end of the cipher, so now we must apply the partial sum techniques of section 2.4. Guessing the four key bytes of the first round first to define our herds and computing the partial sums x_k one at a time allows one to calculate the desired sum with 2^{104} bits of storage and work equivalent to about 2^{202} trial encryptions. We need to do this for about four herds before we start to cut our search tree. (We will need about 26 herds in total to get a unique solution, but the workload is dominated by the first four herds.) The overall attack complexity comes out at 2^{204} encryptions, and thus faster than exhaustive key search for 256-bit keys.

As this attack needs the equivalent of more than 2^{192} encryptions, it seems to be useless for 192-bit keys. But the 192-bit key schedule allows a 2^{16}-fold speed-up for the 8-round attack, which thus requires the equivalent of about $2^{204-16} = 2^{188}$ encryptions. We stress that the time complexity of 2^{188} encryptions only holds for 192-bit keys, not for 256-bit keys.

Each byte of $b^{(5)}$ depends on 21 subkey bytes, namely: all the 16 bytes from $k^{(8)}$, 4 bytes from $k^{(7)'}$ and one byte from $k^{(6)'}$. Similar to Section 2.2, fixing the last round key $k^{(8)}$ determines two of the four bytes from $k^{(7)'}$ and, depending on which byte of $b^{(5)}$ we target, possibly also the relevant subkey byte from $k^{(6)'}$. More precisely, by choosing three columns (12 bytes) of $k^{(8)}$ one can learn two columns of $k^{(7)'}$, and the fourth column of $k^{(8)}$ also determines one column of $k^{(6)'}$. In each column of $k^{(7)'}$ we find one subkey byte we need for the attack. In other words, fixing $k^{(8)'}$ (or even only three columns of $k^{(8)'}$) gives us two useful key bytes of $k^{(7)'}$. (This holds for the 192-bit key schedule. See [Luc00], where the Rijndael key schedule and this weakness are explained in more detail.)

To describe the attack, we look at the partial sums technique from a slightly different point of view. To attack 8-round Rijndael, we check the sum of the values taken on by one byte from $b^{(5)}$ in the 2^{104} encryptions of a herd. For this, we evaluate equation 1 five times: four times on the "bottom level", and, using these four results, a last time taking these four values instead of the ciphertexts $c_{i,0}, \ldots, c_{i,3}$. Each evaluation of equation 1 starts with counters for $(c_0, c_1, c_2, c_3, \langle\text{other}\rangle)$, where the bytes c_i and the corresponding keys are aligned in the same column. It can be described by the following substeps:

1. Guess two key bytes, w.l.o.g. k_0 and k_1, and compute the counters (mod 2) for $(x_{0,1}, c_2, c_3, \langle\text{other}\rangle)$.
2. Guess one key byte, w.l.o.g. k_2, and count $(x_{0,1,2}, c_3, \langle\text{other}\rangle)$.
3. Guess one key byte (k_3), and count $(x_{0,\ldots,3}, \langle\text{other}\rangle)$.

(Note that we just introduced a slightly different notation for the $x_{\langle\text{some}\rangle}$. The reason will become obvious below.)

To attack 8-round Rijndael with 192-bit keys, we obtain $2^{128} - 2^{119}$ texts, guess four first-round subkey bytes to obtain our herds, concentrate on a single herd, and target a single byte in $b^{(5)}$. We continue with guessing three columns of

k^8 and evaluating equation 1 for each column. This gives us gives us 2^{56} counters (mod 2) for

$$(x_{0,\ldots,3}, x_{4,\ldots,7}, x_{8,\ldots,11}, c_{12}, c_{13}, c_{14}, c_{15}).$$

Note that the values $x_{0,\ldots,3}$, $x_{4,\ldots,7}$, and $x_{8,\ldots,11}$ correspond to the bytes c_0, c_1, and c_2 we use within equation 1 to get the final count for the byte of $b^{(5)}$. Now we evaluate (not guess!)[4] two key bytes of $k^{(7)}$ and execute the first substep of the partial sums technique. Essentially for free, we reduce the number of counters from 2^{56} to 2^{48}, and we get counters (mod 2) for

$$(x_{0,\ldots,7}, x_{8,\ldots,11}, c_{12}, c_{13}, c_{14}, c_{15}).$$

By guessing the last column of $k^{(8)}$ and get 2^{24} counters for

$$(x_{0,\ldots,7}, x_{8,\ldots,11}, x_{12,\ldots,15}).$$

Guessing another two bytes of $k^{(7)}$ we get 2^8 counters for x_{15}. We can evaluate one byte of $k^{(6)}$, which allows us to check the balancedness of one byte of $b^{(5)}$.

Note that the order in which things are done is crucial for the time complexity. If we first guessed all 16 bytes from $k^{(8)}$ and only then evaluated the two key bytes from $k^{(7)'}$ which we get for free, the speed-up would only be 2^8 compared to the running time for attacking Rijndael with 256-bit keys. In this case, the time complexity would be 2^{196} for 192-bit keys, i.e. *slower* than exhaustive search.

2.7 Summary

The Square attack can be improved so that it requires 2^{44} work to attack 6 rounds of Rijndael. The extension to 7 rounds has complexity 2^{155} for 192-bit keys and complexity 2^{172} for 256-bit keys. There is also an alternative extension to 7 rounds that can break all key sizes with lower overall complexity (2^{120} work) but which requires virtually the entire codebook of texts ($2^{128} - 2^{119}$ texts). Another result of our analysis is that, for the 256-bit and 192-bit key sizes, one may break 8 rounds of Rijndael faster than by exhaustive search, again with $2^{128} - 2^{119}$ texts. The 256-bit key size requires 2^{204} work, the 192-bit key size 2^{188}.

3 The Key Schedule

Compared to the cipher itself, the Rijndael key schedule appears to be more of an ad hoc design. It has a much slower diffusion structure than the cipher, and contains relatively few non-linear elements.

[4] This is where the 2^{16}-fold speed-up for 192-bit keys comes from.

3.1 Partial Key Guessing

The Rijndael submission document states that the key schedule was designed with the requirement that "Knowledge of a part of the Cipher Key or Round Key bits shall not allow to calculate many other Round Key bits" [DR98, section 7.5]. The key schedule does not seem to achieve that goal.

Let us look at the case of a 128-bit block size and 256-bit key in more detail. The key schedule consists of 8 cycles, which produces a total of 15 round keys (the last half of the last cycle is never used). The key schedule can be seen as four separate rows that only have a limited interaction. We will concentrate on a particular row; say, number i. We guess the values $K_{i,7}^{(s)}$ for $s = 0, \ldots, 6$. (There is no point in guessing it for $s = 7$, as that byte is not used in an expanded key.) Using the recurrent computation rule of the key schedule, we can now compute $K_{i,6}^{(s)}$ for $s = 1, \ldots, 6$, $K_{i,5}^{(s)}$ for $s = 2, \ldots, 6$, etc. (Please refer to appendix A for definitions of our notation, if it is not clear.) All in all, we learn 28 bytes of the expanded key for the cost of having guessed only seven bytes.

The bytes that we guessed in row i are exactly those bytes that affect row $i - 1 \bmod 4$. Thus, if we now guess the first eight bytes of row $i - 1$, then we can compute the rest of that row for a total of 60 bytes, and if we guess a total of 15 bytes, we learn 88 bytes of the expanded key.

We can extend this with further rows, and get 148 bytes of the expanded key by guessing 23 bytes, and 208 bytes by guessing 31 bytes. There are of course many other ways in which guessing some bytes results in knowledge of many more. On a smaller scale, several of our attacks in section 2 used dependencies between round key bytes to reduce the complexity of the attack.

3.2 Key Splitting

Another interesting property is that the key can be "split" into two halves. The two topmost rows interact with the two bottommost rows through only 14 bytes (in the case of 128-bit block size and 256-bit key). If we guess (or know) those 14 bytes, then the rest of the key has been split into two independent halves, each of which controls half of the expanded key bytes. There are many ways to split the key. By rows is the easiest way, but it is also possible to split it by column (at least for a few cycles).

This immediately suggests some kind of meet-in-the-middle attack to a cryptanalyst. However, as the expanded key bytes of the two halves are mixed very thoroughly in the non-linear cipher, we have not found a way to exploit this property. Note that the DES key schedule allows the key bits to be split into 56 independent parts, but no attack is known that uses this property.

3.3 Summary

The fact that these properties are present in spite of the stated design goal is unsettling. Some of our attacks make use of the relations between expanded key bytes and would have a higher complexity if these relations did not exist. The attack on 8-round Rijndael with 192-bit keys would be slower than exhaustive key search without these relations. Our attack in the next section also makes extensive use of the properties of the key schedule.

4 A 9-Round Related-Key Attack

Related-key attacks were first introduced by Biham in [Bih93] and later extended in [KSW96,KSW97]. We assume that the reader is familiar with the basics of related-key cryptanalysis.

The submission states that "The key schedule of Rijndael, with its high diffusion and non-linearity, makes it very improbable that [related-key attacks] can be successful for Rijndael" [DR98, section 8.7], and also lists resistance to related-key attacks as one of the requirements for the Rijndael key schedule [DR98, section 7.5]. We do not feel that the Rijndael key schedule has a very high level of diffusion. It can take many cycles before a low-weight difference starts to affect a significant number of other bytes. This can best be seen if we run the key schedule backwards; each byte affects two other bytes that are (almost) a full cycle further back.

We show how a related-key attack can be mounted on 9 rounds of Rijndael with a 256-bit key. This is basically a variant of the Square attack; we use 256 related keys that differ in a single byte in the fourth round key. We use plaintext differences to cancel out the earlier round key differences, and get three bytes at the end of round 6 that sum to zero when taken over the 256 encryptions. We guess key bytes of the last three rounds to compute backwards from the ciphertext and detect this property.

4.1 The Key Difference Pattern

Starting with an unknown base key L, we derive a set of 256 related keys $L_0, \ldots, L_{255}$. The difference $L_a \oplus L$ takes on the value a in bytes 21 and 25, and is zero elsewhere. The diffusion in the key schedule is slow enough that we can track all the differences in the round keys. Figure 1 shows the difference pattern. The key schedule for the 9-round cipher needs to generate 10 round keys. With a 128-bit block size and a 256-bit key, this requires five cycles of the key schedule, which are shown in the figure. Each of the cycles provides two round keys.

The dark gray bytes are the bytes of L that we guess. The light gray bytes are bytes that we can deduce from the guesses that we have made using the recurrence relationship between the expanded key bytes. We guess a total of 27 bytes of the key, and this allows us to compute a total of 66 bytes of the expanded key. We will use all of our guesses in the attack, but for the moment we concentrate on tracking the differences through the key schedule.

In the first cycle we have a difference a in $K^{(0)}_{1,5}$ and $K^{(0)}_{1,6}$. In the next cycle we get a difference a in $K^{(1)}_{1,5}$. In the third cycle we have difference a in $K^{(2)}_{1,5}$,

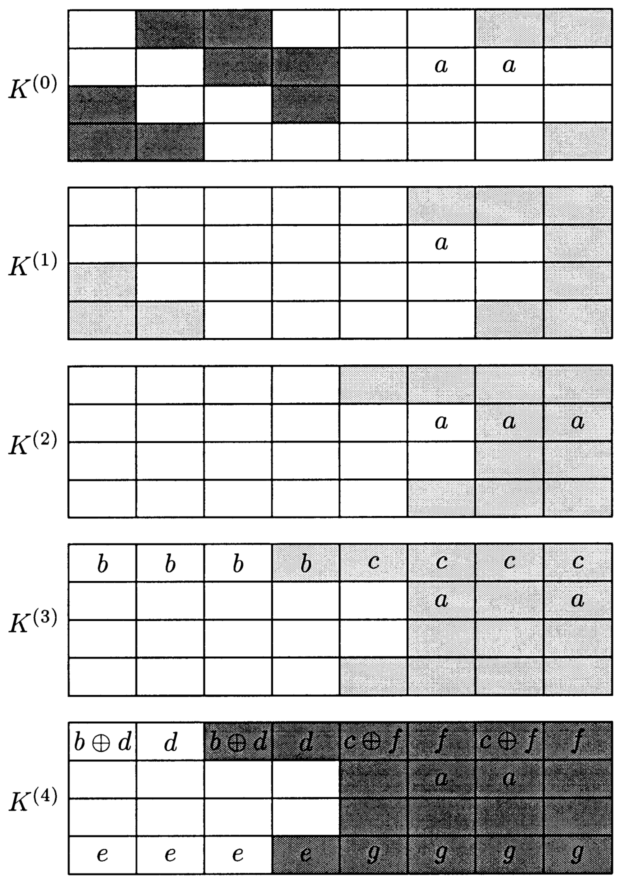


Fig. 1. Difference and guessing pattern in the key of the 9-round attack.

$K^{(2)}_{1,6}$, and $K^{(2)}_{1,7}$. At this point the difference is first confronted with a non-linear S-box. To track the difference, we need to know $K^{(2)}_{1,7}$ of key L; this allows us to compute the output difference b of the S-box given the input difference a. As the shading shows, this key byte can be deduced from the guesses that we have made. In the fourth cycle we get the difference b in $K^{(3)}_{0,i}$ for $i = 0, \ldots, 3$. Again we encounter an S-box, and therefore we need to know $K^{(3)}_{0,3}$ of L. This gives us the output difference c of that S-box given input difference b. We thus get a difference c in $K^{(3)}_{0,i}$ for $i = 4, \ldots, 7$. The differences from the previous cycle also come through as a difference a in $K^{(3)}_{1,5}$ and $K^{(3)}_{1,7}$. We can track the rest of the difference propagation in a similar way as is shown in the figure. All in all, it turns out that we have guessed more than enough bytes of L to be able to track all of the changes. That is, for each value of a, we know the exact value of b, c, d, e, f, and g. Although we are guessing a very large number of bytes that we will use in our attack, we would only need to guess six bytes in order to track the difference pattern through the key schedule.

4.2 The Encryptions

Having guessed the dark-gray bytes shown in figure 1, we encrypt one plaintext under each key. These plaintexts are chosen such that all encryptions end up in the same state after the first round (i.e., after adding the second round key). We know the differences in the second round key $k^{(1)}$ and the key bytes that we guessed allow us to introduce appropriate differences in the plaintexts to ensure the same state after round 1. We now get a single byte difference introduced at the end of round 3; if we look at all our 256 encryptions, this one byte takes on each value exactly once. This propagates to ensure that each byte of $m^{(5)}$ runs over all possible values when taken over the 256 encryptions.

The next few steps are shown in figure 2. The round keys for round 5 and 6 are on the left, with their differences marked. On the right are some of the state differences. The bytes marked O are bytes that take on every possible value exactly once. Bytes marked X can behave in any manner. Bytes marked σ have the property that if you sum them over all 256 encryptions, the sum is zero. The important item to note is that we have three σ bytes in $b^{(6)}$.

We are going to compute $b^{(6)}_{1,3}$ from the ciphertext, our known key bytes, and some additional guessed key bytes. This is shown in the figure with the gray color. Note that we are using an equivalent representation for round 8, where we have swapped the order of the MixColumn and AddRoundKey, and add $k^{(8)\prime}$ instead of $k^{(8)}$. We know the ciphertext and the last round key, so we can compute backwards up to the AddRoundKey of round 8. We now guess the four marked bytes in $k^{(8)\prime}$. (We know several bytes of $k^{(8)}$, but that provides no information about these bytes of $k^{(8)\prime}$. However, as each column of $k^{(8)\prime}$ is the result of an inverse MixColumn operation on $k^{(8)}$, we can propagate our knowledge of the differences from $k^{(8)}$ to $k^{(8)\prime}$.) We can now compute the marked bytes in $t^{(8)}$, $s^{(8)}$, and $t^{(7)}$, and finally we can compute $b^{(6)}_{1,3}$. We check whether this value sums to zero when taken over all 256 encryptions. If this is not the case, we have made a wrong guess somewhere. As before, we can generate enough sets of plaintexts to uniquely identify the correct key guesses that we have made.

All in all, we have guessed 31 bytes of key material, and for each guess we perform an amount of work comparable to a single encryption. This puts the overall complexity of the attack at 2^{248}.

Looking at the plaintext requirements, we do not have to perform 256 encryptions for each of the key byte guesses that we have made. The eight bytes of plaintext that we use to cancel the differences can take on only 2^{64} values, so we can encrypt 2^{64} plaintexts with each of the 256 related keys for a total chosen plaintext requirement of 2^{72}.

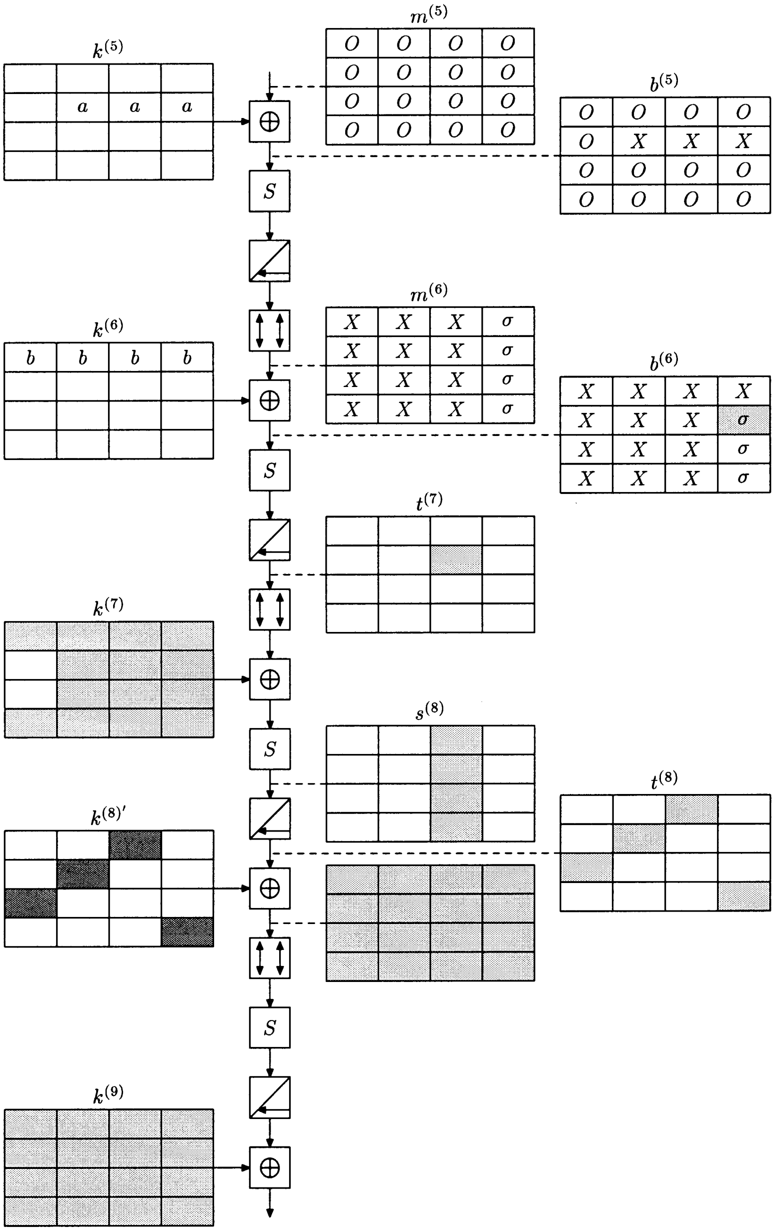


Fig. 2. Rounds 6–9 of the attack.

4.3 An Improvement

We can now use the techniques of section 2.3 to improve this attack. Instead of guessing the eight bytes of the first round key, we use all 2^{64} plaintexts for each of the keys, and sum over all 2^{72} encryptions. We will need about 32 structures of 2^{72} texts overall to uniquely identify the correct key guess, so our plaintext requirement grows to 2^{77}. We now guess the 19 dark gray bytes (152 bits) of $K^{(4)}$ in figure 1. (This also provides the required information to track the differences between the 256 keys.) We decrypt each of the 2^{72} ciphertexts of one structure for one round, and count how often each of the possible values for the four remaining interesting bytes occurs. We are now left with something very similar to equation 1, which requires 2^{48} elementary steps. It is clear that this process is dominated by the work of decrypting the last round. This reduces the attack complexity to $5 \cdot 2^{224}$. (The factor 5 comes from the fact that we need five of these structures before we start cutting into the guesses that dominate the workload.)

4.4 Further Work

There are many ways in which variations on this attack can be made, such as using a different key difference pattern, or possibly applying the partial-sum technique further to reduce the workload. We have not investigated these in any detail. This remains an area for further study.

4.5 Summary

There is a related-key attack on 9 rounds of Rijndael with 256-bit keys that uses 2^{77} plaintexts under 256 related keys, and requires 2^{224} steps to complete.

5 Conclusions

We examined the security of the AES candidate Rijndael, and described several new attacks and unexpected properties of the cipher. Up to now we have only looked at the Rijndael versions with a 128-bit block size. Although similar in structure, Rijndael with larger block sizes is different enough—the byte alignments that are so crucial to some of our attacks are different—that it will have to be analyzed separately.

We introduced the "partial sum" technique, which substantially reduces the workfactor of the dedicated Square attack. We also showed how one may trade texts for time, to penetrate through more rounds of Rijndael when many known texts are available. These techniques allowed us to find attacks that break as many as 7 (of 10) rounds for 128-bit keys, 8 (of 12) rounds for 192-bit keys, and 8 (of 14) rounds for 256-bit keys. Many of these attacks require virtually the entire codebook of texts and hence are not very practical.

The key schedule does not achieve its stated design goals, especially for 192-bit and 256-bit keys. Although we have not found a large-scale exploit of the key schedule properties described in section 3, we find them worrisome.

The 9-round related-key attack has a complexity of 2^{224}, which is of course completely impractical; but it is faster than an exhaustive key search, which is the standard measure to compare against. Our results have no practical significance for anyone using the full Rijndael.

Acknowledgments. The "extended Twofish team" met for two week-long cryptanalysis retreats during fall 1999, once in San Jose and again in San Diego. This paper is a result of those collaborations. Our analysis of Rijndael has very much been a team effort, with everybody commenting on all aspects. Tadayoshi Kohno contributed to our discussions. Ian Goldberg very kindly performed some calculations for us. We would like to thank them both for their contributions and for the great time we had together.

References

Bih93. Eli Biham. New types of cryptanalytic attacks using related keys. In Tor Helleseth, editor, *Advances in Cryptology—EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 398–409. Springer-Verlag, 1993.

DBRP99. Carl D'Halluin, Gert Bijnens, Vincent Rijmen, and Bart Preneel. Attack on six rounds of Crypton. In Lars Knudsen, editor, *Fast Software Encryption '99*, volume 1636 of *Lecture Notes in Computer Science*, pages 46–59. Springer-Verlag, 1999.

DKR97. J. Daemen, L. Knudsen, and V. Rijmen. The block cipher Square. In *Fast Software Encryption '97*, pages 149–165. Springer-Verlag, 1997.

DR98. Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. In *AES Round 1 Technical Evaluation CD-1: Documentation.* NIST, August 1998. See `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/` or `http://www.nist.gov/aes`.

GM00. Henri Gilbert, Marine Minier. A collision attack on 7 rounds of Rijndael. In *The third Advanced Encryption Standard Candidate Conference*, pages 230–241. NIST, April 2000. See `http://www.nist.gov/aes`.

KSW96. John Kelsey, Bruce Schneier, and David Wagner. Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and triple-DES. In Neal Koblitz, editor, *Advances in Cryptology—CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 237–251. Springer-Verlag, 1996.

KSW97. John Kelsey, Bruce Schneier, and David Wagner. Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In *Information and Communications Security, First International Conference Proceedings*, pages 203–207. Springer-Verlag, 1997.

Luc00. Stefan Lucks. Attacking seven rounds of Rijndael under 192-bit and 256-bit keys. In *The third Advanced Encryption Standard Candidate Conference*, pages 215–229. NIST, April 2000. See `http://www.nist.gov/aes`.

A Notation

The original description of Rijndael uses pictures to define the cipher, and re-uses symbols several times. This makes it difficult to refer in an unambiguous manner to intermediate values in an encryption. To help resolve this problem, we define some extra terminology and symbols for various values in the cipher. We have tried to retain as many of the symbols of [DR98] as possible. For completeness, we have named every intermediate value that seemed of use to us. Many of these definitions are not used in this paper but are included for completeness. All our explanations of the symbols refer to the description in [DR98].

3a	The byte 3a_{16} (and similarly for all other byte values). This can either be a direct byte value, or it can be interpreted as an element of GF(2^8).
$a_{i,j}^{(r)}$	The byte at position (i, j) at the beginning of round r.
$b_{i,j}^{(r)}$	The byte at position (i, j) at the output of round r (just after the key addition).
$c(x)$	The polynomial $\mathsf{03}x^3 + \mathsf{01}x^2 + \mathsf{01}x + \mathsf{02}$ that is used to define the MDS matrix.
c_i	The bytes of the ciphertext, where $i \in \{0, \ldots, 4N_b - 1\}$.
C_i	The number of positions that row i is shifted left in the ShiftRow function.
$k_{i,j}^{(r)}$	The expanded key byte in round r at position (i, j) where $r \in \{0, \ldots, N_r\}$, $i \in \{0, \ldots, 3\}$ and $j \in \{0, \ldots, N_b - 1\}$. For $r = 0$, it is the key that is XORed into the state before the first round. The entire round key is referred to as $k^{(r)}$.
$k_{i,j}^{(r)'}$	This is a simple linear function of the round key $k^{(r)}$. XORing $k^{(r)'}$ into the state before the MixColumn operation is equivalent to XORing $k^{(r)}$ into the state after the MixColumn operation (when looking at encryption).
K_i	The bytes of the expanded key in their canonical order, where $i \in \{0, \ldots, 4N_b(N_r + 1) - 1\}$. Note that the bytes $K_0, \ldots, K_{4N_k - 1}$ form the key of the cipher itself.
$K_{i,j}^{(s)}$	The expanded key bytes in cycle s at position (i, j), where $s \in \{0, \ldots, N_s - 1\}$, $i \in \{0, \ldots, 3\}$, and $j \in \{0, \ldots, N_k - 1\}$.
$m_{i,j}^{(r)}$	The byte at position (i, j) at the output of the MixColumn operation in round r.
M	The MDS matrix.
N_b	The block size (in bits) divided by 32.
N_k	The number of key bits divided by 32.
N_r	The number of rounds.
N_s	The number of cycles in the key expansion; $N_s = \lceil (N_r + 1)N_b/N_k \rceil$.
p_i	The bytes of the plaintext, where $i \in \{0, \ldots, 4N_b - 1\}$.
r	The round number. The rounds are numbered $1, \ldots, N_r$, and the value 0 is sometimes used to refer to the initial AddRoundKey operation.
$R_i^{(s)}$	The round constant used at position $(i, 0)$ in cycle s.
s	The cycle number in the key expansion. Each cycle produces $4N_k$ expanded key bytes. The cycles are numbered from 0 to $N_s - 1$.
$s_{i,j}^{(r)}$	The byte at position (i, j) at the output of the S-boxes in round r.
S	The S-box. Entry x is written as $S[x]$. The inverse S-box is written as S^{-1}.

$t_{i,j}^{(r)}$	The byte at position (i,j) at the output of the ShiftRow operation in round r.

Note that the key schedule operates in what we call "cycles." This is to distinguish it from the rounds of the cipher itself. If the block size and key size are the same, then a cycle corresponds to a round, but this is not the case in general.

We can now give the various formulae through which these values are tied together. This provides a complete specification of the cipher, although not one that is easy to understand. Note that N_b and N_k are the cipher parameters that can each take on the values 4, 6, and 8. The multiplication of two bytes is defined as multiplication in $\mathrm{GF}(2)[x]/(x^8+x^4+x^3+x+1)$ and the byte value $\sum_{i=0}^{7} a_i 2^i$ with $a_i \in \mathrm{GF}(2)$ is identified with the field element $\sum_{i=0}^{7} a_i x^i$.

$$\begin{aligned}
a_{i,j}^{(1)} &= p_{4j+i} \oplus k_{i,j}^{(0)} && \text{Initial key addition}\\
s_{i,j}^{(r)} &= S[a_{i,j}^{(r)}] && \text{ByteSub}\\
t_{i,j}^{(r)} &= s_{i,(j+C_i) \bmod N_b}^{(r)} && \text{ShiftRow}\\
[m_{0,j}^{(r)}, \ldots, m_{3,j}^{(r)}]^T &= M[t_{0,j}^{(r)}, \ldots, t_{3,j}^{(r)}]^T && \text{MixColumn}\\
b_{i,j}^{(r)} &= m_{i,j}^{(r)} \oplus k_{i,j}^{(r)} && \text{AddRoundKey}\\
a_{i,j}^{(r)} &= b_{i,j}^{(r-1)} && \text{for } r = 2, \ldots, N_r\\
c_{4j+i} &= t_{i,j}^{(N_r)} \oplus k_{i,j}^{(N_r)} && \text{Final round}\\
k_{i,j}^{(r)} &= K_{4rN_b+4j+i} && \text{Round keys}\\
K_{4sN_k+4j+i} &= K_{i,j}^{(s)} &&\\
K_{i,0}^{(s)} &= K_{i,0}^{(s-1)} \oplus R_i^{(s)} \oplus &&\\
&\qquad S[K_{(i+1) \bmod 4, N_k-1}^{(s-1)}] && \text{for } s = 1, \ldots, N_s - 1\\
K_{i,4}^{(s)} &= S[K_{i,3}^{(s)}] \oplus K_{i,j}^{(s-1)} && \text{if } N_k = 8\\
K_{i,j}^{(s)} &= K_{i,j-1}^{(s)} \oplus K_{i,j}^{(s-1)} && \text{if } j > 0 \text{ and } (j \neq 4 \vee N_k \neq 8)\\
C_i &= i + \lfloor i/2 \rfloor \cdot \lfloor N_b/8 \rfloor && \text{for } i = 0, \ldots, 3\\
N_r &= \max(N_b, N_k) + 6 && \text{Number of rounds}\\
R_i^{(s)} &= 0 && \text{for } i > 0\\
R_0^{(s)} &= \text{field element } x^{s-1} &&\\
M &= \begin{pmatrix} \mathsf{02\ 03\ 01\ 01} \\ \mathsf{01\ 02\ 03\ 01} \\ \mathsf{01\ 01\ 02\ 03} \\ \mathsf{03\ 01\ 01\ 02} \end{pmatrix} &&
\end{aligned}$$

On the Pseudorandomness of the AES Finalists – RC6 and Serpent

Tetsu Iwata and Kaoru Kurosawa

Department of Electrical and Electronic Engineering,
Faculty of Engineering,
Tokyo Institute of Technology
2–12–1 O-okayama, Meguro-ku, Tokyo 152–8552, Japan
`tez@ss.titech.ac.jp`, `kurosawa@ss.titech.ac.jp`

Abstract. Luby and Rackoff idealized DES by replacing each *round function* with *one large* random function. In this paper, we introduce a primitive-wise idealization in which some of the *primitive operations* of the round function are left untouched and some of them are replaced with *small* random functions or permutations. We then prove that a four round primitive-wise idealized RC6 is not a pseudorandom permutation and a three round primitive-wise idealized Serpent is a super-pseudorandom permutation.

1 Introduction

There are five AES finalists, RC6, Serpent, MARS, Twofish and Rijndael. RC6 was proposed by Rivest et.al. [6] as a successor of RC5. RC6 makes essential use of data-dependent rotations in the new structure. It also includes the use of four working registers and the inclusion of integer multiplication as an additional primitive operation. Serpent was proposed by Anderson et.al. [1]. Each round of Serpent has 32 parallel S-boxes and a following linear transformation of 128 bits. MARS was proposed by Burwick et.al. [2]. It uses a so called type-3 Feistel structure. Twofish was proposed by Schneier et.al. [7]. It has a 16 round Feistel structure. Rijndael was proposed by Daemen et.al. [3]. Its round transformation consists of three distinct invertible uniform transformations.

We consider the security of block ciphers in two ways, pseudorandomness and super-pseudorandomness.

– Pseudorandomness means that no attacker with polynomially many *encryption* queries can distinguish between the block cipher and a truly random permutation. This security corresponds to a chosen *plaintext* attack.
– Super-pseudorandomness means that no attacker with polynomially many *encryption and decryption* queries can distinguish between the block cipher and a truly random permutation. This security corresponds to a chosen *plaintext and ciphertext* attack.

Note that super-pseudorandomness implies pseudorandomness.

Luby and Rackoff idealized DES by replacing each *round function* with *one large* random function. Then they showed that the idealized three round DES yields a pseudorandom permutation and the idealized four round DES yields a super-pseudorandom permutation [4]. Maurer gave a simpler proof for non-adaptive adversaries [5].

For this kind of idealization, the three round idealized Twofish is a pseudorandom permutation and the four round idealized Twofish is a super-pseudorandom permutation because Twofish has the same Feistel structure as DES. MARS has a so called type-3 Feistel structure. At the rump session of AES2, Vaudenay and Moriai claimed that the five round idealized MARS is a pseudorandom permutation [8].

In this paper, we introduce a primitive-wise idealization in which some of the *primitive operations* of the round function (e.g., linear transformations and etc.) are left untouched and some of them (e.g., S-boxes and etc.) are replaced with *small* random functions or permutations. It is not known whether such a primitive-wise idealized DES is pseudorandom (or super-pseudorandom). Similarly, the same problem is open for all the AES candidates.

We solve this problem for RC6 partially, and solve for Serpent. We first idealize RC6 by replacing only an "$x \times (2x+1)$"operation with a pseudorandom function. The data-dependent rotation parts and the connections among the four registers are left untouched because they are the main properties of RC6. We then prove that the four round primitive-wise idealized RC6 is not a pseudorandom permutation for non-adaptive adversaries.

Serpent is idealized similarly. The linear transformation parts are left untouched and only the S-boxes are replaced with small pseudorandom permutations. We then prove that the two round primitive-wise idealized Serpent is not a pseudorandom permutation and the three round primitive-wise idealized Serpent is a super-pseudorandom permutation for non-adaptive adversaries.

A similar analysis for Rijndael, MARS, and Twofish is now in progress. Our results are stronger than the previous results for DES, Twofish [4] and MARS [8] because our idealization assumes weaker and smaller modifications of the ciphers.

This paper is organized as follows. In Section 2, we review the security model and the pseudorandomness of Twofish and MARS. The primitive-wise idealized RC6 is studied in Section 3 and the primitive-wise idealized Serpent is studied in Section 4.

2 Preliminaries

2.1 Security Model

Let us consider a computationally unbounded distinguisher $\mathcal{A}$ with an oracle $\mathcal{O}$. The oracle $\mathcal{O}$ chooses a permutation π randomly from the set of all permutations C^* over $\{0,1\}^n$ or from a subset of permutations $C \subset C^*$ (For a block cipher, C is the set of permutations obtained from all the keys). The aim of the distinguisher $\mathcal{A}$ is to distinguish if the oracle $\mathcal{O}$ implements C^* or C. Let p_{C^*} denote the probability that $\mathcal{A}$ outputs 1 when $\mathcal{O}$ implements C^* and p_C denote the probability that $\mathcal{A}$ outputs 1 when $\mathcal{O}$ implements C. That is,

$$p_{C^*} \overset{\triangle}{=} \Pr(\mathcal{A} \text{ outputs } 1 \mid \mathcal{O} \leftarrow C^*) \text{ and } p_C \overset{\triangle}{=} \Pr(\mathcal{A} \text{ outputs } 1 \mid \mathcal{O} \leftarrow C) \ .$$

Then the advantage $\mathtt{Adv}_{\mathcal{A}}$ of the distinguisher $\mathcal{A}$ is defined as

$$\mathtt{Adv}_{\mathcal{A}} \overset{\triangle}{=} |p_C - p_{C^*}| \ .$$

Suppose that $\mathcal{A}$ is limited to make at most $poly(n)$ queries to $\mathcal{O}$, where $poly(n)$ is some polynomial in n. We say that $\mathcal{A}$ is a pseudorandom distinguisher if it queries x and the oracle answers $y = \pi(x)$, where π is a randomly chosen permutation by $\mathcal{O}$. We say that $\mathcal{A}$ is a super-pseudorandom distinguisher if it is also allowed to query y and receives $x = \pi^{-1}(y)$ from the oracle.

Finally, C is called a pseudorandom permutation ensemble if $\mathtt{Adv}_{\mathcal{A}}$ is negligible for any pseudorandom distinguisher (A pseudorandom function ensemble is defined similarly). C is called a super-pseudorandom permutation ensemble if $\mathtt{Adv}_{\mathcal{A}}$ is negligible for any super-pseudorandom distinguisher. On the other hand, C^* is called the truly random permutation ensemble.

In this paper, we consider a non-adaptive distinguisher, i.e., a distinguisher that sends all the queries to the oracle at the same time.

2.2 Pseudorandomness of Idealized Twofish

Twofish has the same Feistel structure as DES shown in Fig. 1.

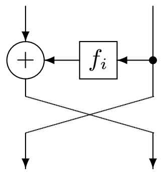


Fig. 1. The i-th round of the idealized Twofish

Assume that each round functions f_i is an independent pseudorandom function from $\{0,1\}^{n/2}$ to $\{0,1\}^{n/2}$. Then the following propositions are derived from the result of Luby and Rackoff [4].

Proposition 1. *The four round idealized Twofish is a super-pseudorandom permutation.*

Proposition 2. *The three round idealized Twofish is a pseudorandom permutation.*

2.3 Pseudorandomness of Idealized MARS

MARS has a structure as shown in Fig. 2.

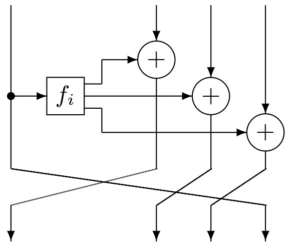


Fig. 2. The i-th round of the idealized MARS

Assume that each round function f_i is an independent pseudorandom function from $\{0,1\}^{n/4}$ to $\{0,1\}^{3n/4}$. Then Vaudenay and Moriai claimed the following proposition [8].

Proposition 3. *The five round idealized MARS is a pseudorandom permutation.*

3 Pseudorandomness of Primitive-Wise Idealized RC6

3.1 Primitive-Wise Idealization of RC6

RC6 is specified as RC6-$w/r/b$, where w denotes the number of bits of a word, r denotes the number of rounds, and b denotes the length of the encryption key in bytes. RC6 works with four w bits registers, A, B, C, and D. The i-th round of RC6 is defined as follows.

$$\begin{aligned}
&t = (B \times (2B+1)) \lll \lg w \\
&u = (D \times (2D+1)) \lll \lg w \\
&A = ((A \oplus t) \lll u) + S[2i] \\
&C = ((C \oplus u) \lll t) + S[2i+1] \\
&(A, B, C, D) = (B, C, D, A)
\end{aligned}$$

Definition of the i-th round of RC6

In the above definition, $a+b$ is an addition modulo 2^w, $a \oplus b$ is a bitwise exclusive-or of two w bits words, $a \times b$ is a multiplication modulo 2^w and $a \lll b$ denotes to rotate a w bits word a to the left by x, where x is the number given by the least significant $\lg w$ bits of b and $\lg w$ denotes the base-two logarithm of w. Finally, $S[2i]$ and $S[2i+1]$ denote the i-th round key.

Let n denote the length of a plaintext. Then $n = 4w$. In other words, each of A, B, C, D takes an element of $\{0,1\}^{n/4}$.

Now we idealize RC6 as shown below, where each f_j is an independent pseudorandom function from $\{0,1\}^{n/4}$ to $\{0,1\}^{n/4}$.

$$\begin{aligned}
&t = f_{2i}(B)\\
&u = f_{2i+1}(D)\\
&A = ((A \oplus t) \lll u)\\
&C = ((C \oplus u) \lll t)\\
&(A, B, C, D) = (B, C, D, A)
\end{aligned}$$

The i-th round of the primitive-wise idealized RC6

Note that

1. We replace t and $S[2i]$ with f_{2i}, and u and $S[2i+1]$ with f_{2i+1}.
2. However, we leave the data-dependent rotations $\lll t$, $\lll u$ and the connections among the four registers untouched because they are the main properties of RC6.

3.2 Pseudorandomness of Primitive-Wise Idealized RC6

The primitive-wise idealized RC6 is illustrated in Fig. 3, where $x = (x_0, x_1, x_2, x_3)$ denotes a plaintext, $z = (z_0, z_1, z_2, z_3)$ and $w = (w_0, w_1, w_2, w_3)$ denote ciphertexts of the three and four round primitive-wise idealized RC6, respectively. Each of x_i, z_i, and w_i is $n/4$ bits long.

Theorem 1. *The four round primitive-wise idealized RC6 is not a pseudorandom permutation.*

Proof. Let C be the set of permutations over $\{0,1\}^n$ obtained from the four round primitive-wise idealized RC6. We consider a distinguisher $\mathcal{A}$ such as follows.

1. $\mathcal{A}$ randomly chooses two plaintexts $x^{(1)} = (x_0^{(1)}, x_1^{(1)}, x_2^{(1)}, x_3^{(1)})$ and $x^{(2)} = (x_0^{(2)}, x_1^{(2)}, x_2^{(2)}, x_3^{(2)})$ such that
$$x_0^{(1)} \neq x_0^{(2)} \text{ and } x_1^{(1)} = x_1^{(2)}, x_2^{(1)} = x_2^{(2)}, x_3^{(1)} = x_3^{(2)} \ . \tag{1}$$
2. $\mathcal{A}$ sends them to the oracle and receives the ciphertexts $w^{(1)} = (w_0^{(1)}, w_1^{(1)}, w_2^{(1)}, w_3^{(1)})$ and $w^{(2)} = (w_0^{(2)}, w_1^{(2)}, w_2^{(2)}, w_3^{(2)})$ from the oracle.
3. Finally, $\mathcal{A}$ outputs 1 if and only if
$$((w_0^{(1)} \oplus w_0^{(2)}) \lll l) = x_0^{(1)} \oplus x_0^{(2)} \tag{2}$$
for some $0 \le l < n/4$.

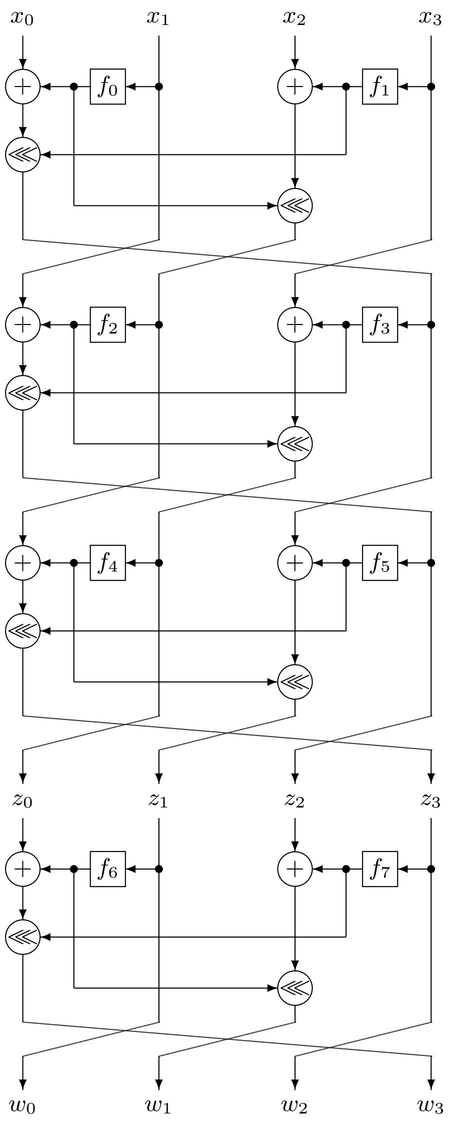


Fig. 3. The primitive-wise idealized RC6

Suppose that the oracle implements the truly random permutation ensemble C^*. Then for any fixed $x^{(1)}$ and $x^{(2)}$ satisfying (1),

$$\Pr(\mathcal{A}\text{ outputs }1) = \frac{\#\{(w_0^{(1)}, w_0^{(2)}) \mid \text{eq.(2) holds for some } l\}}{\#\{(w_0^{(1)}, w_0^{(2)})\}} .$$

It is clear that

$$\#\{(w_0^{(1)}, w_0^{(2)})\} = (2^{n/4})^2 = 2^{n/2} .$$

For each $w_0^{(1)}$ and l, there exists a unique $w_0^{(2)}$ which satisfies eq.(2). Therefore,

$$\#\{(w_0^{(1)}, w_0^{(2)}) \mid \text{eq.(2) holds for some } l\} \le \frac{n}{4} \times 2^{n/4} .$$

Hence,

$$\Pr(\mathcal{A}\text{ outputs }1) \le \frac{n/4 \times 2^{n/4}}{2^{n/2}} = \frac{n}{4 \times 2^{n/4}} .$$

Consequently,

$$p_{C^*} = E_{x^{(1)},x^{(2)}}(\Pr(\mathcal{A}\text{ outputs }1)) \le \frac{n}{4 \times 2^{n/4}} .$$

Next suppose that the oracle implements the four round primitive-wise idealized RC6. We first assume that each f_i is a truly random function. Define α_1, β_1, δ_1 and γ_1 as shown in Fig. 4,

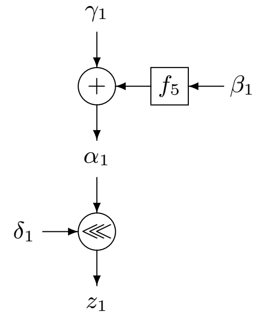


Fig. 4. 3-rd branch in the 3-rd round

Fix $x^{(1)} = (x_0^{(1)}, x_1^{(1)}, x_2^{(1)}, x_3^{(1)})$ and $x^{(2)} = (x_0^{(2)}, x_1^{(2)}, x_2^{(2)}, x_3^{(2)})$ such that $x_0^{(1)} \neq x_0^{(2)}$, $x_1^{(1)} = x_1^{(2)}$, $x_2^{(1)} = x_2^{(2)}$ and $x_3^{(1)} = x_3^{(2)}$ arbitrarily. Then

$$\gamma_1^{(1)} \oplus \gamma_1^{(2)} = ((x_0^{(1)} \oplus x_0^{(2)}) \lll l')$$

for some l' since $x_1^{(1)} = x_1^{(2)}$, $x_2^{(1)} = x_2^{(2)}$ and $x_3^{(1)} = x_3^{(2)}$. If $\beta_1^{(1)} = \beta_1^{(2)}$, then $\alpha_1^{(1)} \oplus \alpha_1^{(2)} = \gamma_1^{(1)} \oplus \gamma_1^{(2)}$. Thus,

$$\alpha_1^{(1)} \oplus \alpha_1^{(2)} = ((x_0^{(1)} \oplus x_0^{(2)}) \lll l') .$$

Further if $\delta_1^{(1)} = \delta_1^{(2)}$, then

$$z_1^{(1)} \oplus z_1^{(2)} = ((\alpha_1^{(1)} \oplus \alpha_1^{(2)}) \lll l'')$$

for some l''. Therefore,

$$z_1^{(1)} \oplus z_1^{(2)} = ((x_0^{(1)} \oplus x_0^{(2)}) \lll l''')$$

for some l'''. Hence, if both $\beta_1^{(1)} = \beta_1^{(2)}$ and $\delta_1^{(1)} = \delta_1^{(2)}$ occur, then

$$((w_0^{(1)} \oplus w_0^{(2)}) \lll l) = x_0^{(1)} \oplus x_0^{(2)}$$

holds for some l because $z_1^{(1)} = w_0^{(1)}$ and $z_1^{(2)} = w_0^{(2)}$. Therefore,

$$p_C \geq \Pr(\beta_1^{(1)} = \beta_1^{(2)} \text{ and } \delta_1^{(1)} = \delta_1^{(2)}) \ .$$

Since $0 \leq l < n/4$ and f_4 is a truly random function, it is easy to see that

$$\Pr(\delta_1^{(1)} = \delta_1^{(2)}) \geq \frac{1}{n/4} \ .$$

Since $x_1^{(1)} = x_1^{(2)}$ and the output of f_2 for $x^{(1)}$ is equal to that for $x^{(2)}$,

$$\Pr(\beta_1^{(1)} = \beta_1^{(2)}) \geq \frac{1}{n/4} \ .$$

Further, β_1 and δ_1 are independent because f_4 is a truly random function. Consequently,

$$\begin{aligned} p_C &\geq \Pr(\beta_1^{(1)} = \beta_1^{(2)}) \times \Pr(\delta_1^{(1)} = \delta_1^{(2)}) \\ &\geq \frac{1}{n/4} \times \frac{1}{n/4} \\ &= \frac{16}{n^2} \ . \end{aligned}$$

Therefore, we obtain that

$$\mathtt{Adv}_{\mathcal{A}} = |p_C - p_{C^*}| \geq \frac{16}{n^2} - \frac{n}{4 \times 2^{n/4}} \ ,$$

which is non-negligible. Finally, we can show that $\mathtt{Adv}_{\mathcal{A}}$ is non-negligible even if each f_i is a pseudorandom function. The proof is almost the same as the proof of [4, Theorem 1]. Hence, the four round primitive-wise idealized RC6 is not a pseudorandom permutation. □

The above theorem implies that the four round primitive-wise idealized RC6 is not a super-pseudorandom permutation.

4 Pseudorandomness of Primitive-Wise Idealized Serpent

4.1 Primitive-Wise Idealization of Serpent

Serpent consists of 32 rounds. The plaintext becomes the first intermediate data B_0, after which the 32 rounds are applied, where each round i consists of three operations:

1. Key Mixing: At each round, a 128 bits subkey K_i is exclusive or'ed with the current intermediate data B_i.
2. S-Boxes: The 128 bits combination of input and key is considered as four 32 bits words. The S-box is applied to these four words, and the result is four output words. The CPU is employed to execute the 32 copies of the S-box simultaneously, resulting with $\mathcal{S}_i(B_i, K_i)$. Each S-box is a permutation over $\{0,1\}^4$.
3. Linear Transformation: The 32-bit in each of the output words are linearly mixed, by

$$\begin{aligned}
X_0, X_1, X_2, X_3 &:= \mathcal{S}_i(B_i, K_i) \\
X_0 &:= X_0 \lll 13 \\
X_2 &:= X_2 \lll 3 \\
X_1 &:= X_1 \oplus X_0 \oplus X_2 \\
X_3 &:= X_3 \oplus X_2 \oplus (X_0 \ll 3) \\
X_1 &:= X_1 \lll 1 \\
X_3 &:= X_3 \lll 7 \\
X_0 &:= X_0 \oplus X_1 \oplus X_3 \\
X_2 &:= X_2 \oplus X_3 \oplus (X_1 \ll 7) \\
X_0 &:= X_0 \lll 5 \\
X_2 &:= X_2 \lll 22 \\
B_{i+1} &:= X_0, X_1, X_2, X_3 \ ,
\end{aligned}$$

   where $\lll$ denotes rotation, and $\ll$ denotes shift.

The effect of the linear transformation is that each plaintext bit affects all the data bits after three rounds. This can be detailed as follows. 4 output bits of some S-box in the first round are expanded by the linear transformation, so that they are input bits to m S-boxes in the second round. Then the $4m$ output bits of these m S-boxes are expanded so that they become input bits to the 32 S-boxes in the third round. The maximum value of m is 19, and the minimum is 17.

We idealize Serpent as shown in Fig. 5 and:

1. Let $n = 128 \times k$ denote the length of a plaintext.

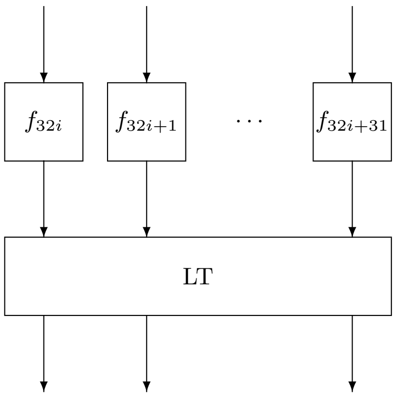


Fig. 5. The i-th round of the primitive-wise idealized Serpent

2. We assume that each f_i is an independent pseudorandom permutation over $\{0,1\}^{4k}$.
3. In the linear transformation, $a \lll b$ is replaced with $a \lll bk$, and $a \ll b$ is replaced with $a \ll bk$.

Note that we leave the linear transformation part untouched except the above modification.

4.2 Pseudorandomness of Primitive-Wise Idealized Serpent

The three round primitive-wise idealized Serpent is illustrated in Fig. 6. Let $x = (x_0, \ldots, x_{31})$ denote a plaintext, $z = (z_0, \ldots, z_{31})$ and $y = (y_0, \ldots, y_{31})$ denote ciphertexts of the two round and the three round primitive-wise idealized Serpent, respectively. Each of x_i, z_i, and y_i is $4k$ bits long.

We first prove the following theorem.

Theorem 2. *The two round primitive-wise idealized Serpent is not a pseudorandom permutation.*

Proof. Let C be the set of permutations over $\{0,1\}^n$ obtained from the two round primitive-wise idealized Serpent. We consider a distinguisher $\mathcal{A}$ such as follows.

1. $\mathcal{A}$ chooses two plaintexts, $x^{(1)} = (x_0^{(1)}, \ldots, x_{31}^{(1)})$ and $x^{(2)} = (x_0^{(2)}, \ldots, x_{31}^{(2)})$ such that $x_0^{(1)} \neq x_0^{(2)}$ and $x_1^{(1)} = x_1^{(2)}, \ldots, x_{31}^{(1)} = x_{31}^{(2)}$.
2. $\mathcal{A}$ sends them to the oracle and receives the ciphertexts $z^{(1)} = (z_0^{(1)}, \ldots, z_{31}^{(1)})$ and $z^{(2)} = (z_0^{(2)}, \ldots, z_{31}^{(2)})$ from the oracle.
3. $\mathcal{A}$ computes $v^{(1)} = \mathrm{LT}^{-1}(z^{(1)})$ and $v^{(2)} = \mathrm{LT}^{-1}(z^{(2)})$.

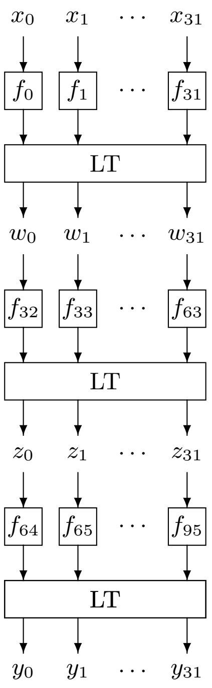

x_0 x_1 $\cdots$ x_{31}
f_0 f_1 $\cdots$ f_{31}
LT
w_0 w_1 $\cdots$ w_{31}
f_{32} f_{33} $\cdots$ f_{63}
LT
z_0 z_1 $\cdots$ z_{31}
f_{64} f_{65} $\cdots$ f_{95}
LT
y_0 y_1 $\cdots$ y_{31}

Fig. 6. The primitive-wise idealized Serpent

4. $\mathcal{A}$ outputs 1 if and only if $v_1^{(1)} = v_1^{(2)}$.

Suppose that the oracle implements the truly random permutation ensemble C^*. Then it is clear that $p_{C^*} = 1/2^{4k}$.

Next suppose that the oracle implements the two round primitive-wise idealized Serpent. The input to f_{33} includes no output of f_0. Therefore, $v_1^{(1)} = v_1^{(2)}$ because $x_1^{(1)} = x_1^{(2)}, \ldots, x_{31}^{(1)} = x_{31}^{(2)}$. Hence $p_C = 1$.

Therefore

$$\texttt{Adv}_{\mathcal{A}} = |p_C - p_{C^*}| = 1 - \frac{1}{2^{4k}} \ .$$

Consequently, $\texttt{Adv}_{\mathcal{A}}$ is non-negligible. Hence, the two round primitive-wise idealized Serpent is not a pseudorandom permutation. □

The above theorem implies that the two round primitive-wise idealized Serpent is not a super-pseudorandom permutation.

We next prove the following theorem.

Theorem 3. *The three round primitive-wise idealized Serpent is a pseudorandom permutation for non-adaptive adversaries.*

Proof. Let C be the set of permutations over $\{0,1\}^n$ obtained from the three round primitive-wise idealized Serpent. First, assume that each f_i is a truly random permutation.

Suppose that $\mathcal{A}$ makes p oracle calls. In the i-th oracle call, $\mathcal{A}$ sends a plaintext $x^{(i)} = (x_0^{(i)}, \ldots, x_{31}^{(i)})$ to the oracle and receives the ciphertext $y^{(i)} = (y_0^{(i)}, \ldots, y_{31}^{(i)})$ from the oracle. In Fig. 6, let $w^{(i)} = (w_0^{(i)}, \ldots, w_{31}^{(i)})$ denote the inputs to $f_{32}, \ldots, f_{63}$ and $z^{(i)} = (z_0^{(i)}, \ldots, z_{31}^{(i)})$ denote the inputs to $f_{64}, \ldots, f_{95}$.

Without loss of generality, we can assume that $x^{(1)}, \ldots, x^{(p)}$ are all distinct. Let $\mathcal{E}_{z_t}$ be the event that $z_t^{(1)}, \ldots, z_t^{(p)}$ are all distinct for $t = 0, \ldots, 31$, and let $\mathcal{E}_z$ be the event that all $\mathcal{E}_{z_0}, \ldots, \mathcal{E}_{z_{31}}$ occur. If $\mathcal{E}_z$ occurs, then, $y^{(1)}, \ldots, y^{(p)}$ are completely random since $f_{64}, \ldots, f_{95}$ are truly random permutations. Therefore, $\mathtt{Adv}_{\mathcal{A}}$ is upper bounded by

$$\mathtt{Adv}_{\mathcal{A}} = |p_C - p_{C^*}| \le 1 - \Pr(\mathcal{E}_z) \ .$$

Further, it is easy to see that

$$1 - \Pr(\mathcal{E}_z) \le \sum_{1 \le i < j \le p} \Pr(z_0^{(i)} = z_0^{(j)}) + \cdots + \sum_{1 \le i < j \le p} \Pr(z_{31}^{(i)} = z_{31}^{(j)}) \ . \qquad (3)$$

Fix $i \ne j$ arbitrarily. We show that all $\Pr(z_0^{(i)} = z_0^{(j)}), \ldots, \Pr(z_{31}^{(i)} = z_{31}^{(j)})$ are sufficiently small. Since $x^{(i)} \ne x^{(j)}$, we have $x_s^{(i)} \ne x_s^{(j)}$ for some $0 \le s \le 31$. For this s, f_s has $4k$ output bits. From the property of LT, the output bits of f_s are distributed among m w_t's, say $t = t_0, \ldots, t_{m-1}$, where m depends of s. Each w_t contains at least k bits of those from our modification of LT. Therefore,

$$\Pr(w_t^{(i)} = w_t^{(j)}) \le \frac{1}{2^k}$$

for $t = t_0, \ldots, t_{m-1}$ because f_s is a truly random permutation.

Next each w_t becomes the input to f_{32+t}. The output bits of $f_{32+t_0}, \ldots, f_{32+t_{m-1}}$ are distributed among all of $z_0, \ldots, z_{31}$ from the property of LT. Each z_u contains at least k bits of those from our modification of LT.

Let $\mathcal{E}_w$ be the event that $w_t^{(i)} \ne w_t^{(j)}$ for $t = t_0, \ldots, t_{m-1}$. Then we have

$$\begin{aligned}\Pr(z_u^{(i)} = z_u^{(j)}) &\le \frac{1}{2^k} \Pr(\mathcal{E}_w) + (1 - \Pr(\mathcal{E}_w)) \\ &\le \frac{1}{2^k} + \Pr(w_{t_0}^{(i)} = w_{t_0}^{(j)}) + \cdots + \Pr(w_{t_{m-1}}^{(i)} = w_{t_{m-1}}^{(j)}) \\ &\le \frac{1}{2^k} + \frac{m}{2^k}\end{aligned}$$

for $u = 0, \ldots, 31$. Therefore, the right side of (3) is upper bounded as follows.

$$\begin{aligned}\sum_{1 \le i < j \le p} \Pr(z_0^{(i)} = z_0^{(j)}) + \cdots + \sum_{1 \le i < j \le p} \Pr(z_{31}^{(i)} = z_{31}^{(j)}) &\le \frac{16(m+1)p^2}{2^k} \\ &\le \frac{320 \times p^2}{2^{n/128}} \ ,\end{aligned}$$

because $m \le 19$ and $n = 128 \times k$.

Since $p = poly(n)$, $\mathtt{Adv}_{\mathcal{A}}$ is negligible for any $\mathcal{A}$. Finally, we can show that $\mathtt{Adv}_{\mathcal{A}}$ is negligible even if each f_i is a pseudorandom permutation as the proof of [4, Theorem 1]. □

We can prove the following corollary similarly.

Corollary 1. *The three round primitive-wise idealized Serpent is a super-pseudorandom permutation for non-adaptive adversaries.*

References

1. R.Anderson, E.Biham and L.Knudsen. *Serpent: a proposal for the Advanced Encryption Standard*. AES proposal, available on: `http://www.cl.cam.ac.uk/~rja14/serpent.html`.
2. C.Burwick, D.Coppersmith, E.D'Avignon, R.Gennaro, S.Halevi, C.Jutla, S.M.Matyas Jr., L.O'Connor, M.Peyravian, D.Safford and N.Zunic. *MARS — a candidate cipher for AES*. AES proposal, available on: `http://www.research.ibm.com/security/mars.html`.
3. J.Daemen and V.Rijmen. *AES proposal: Rijndael*. AES proposal, available on: `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/`.
4. M.Luby and C.Rackoff. *How to construct pseudorandom permutations from pseudorandom functions*. SIAM Journal on Computing, volume 17, number 2, pages 373–386, April 1988.
5. U.M.Maurer. *A simplified and generalized treatment of Luby-Rackoff pseudorandom permutation generators*. Advances in Cryptology — Eurocrypt '92, Lecture Notes in Computer Science, volume 658, pages 239–255, Springer-Verlag, 1992.
6. R.L.Rivest, M.J.B.Robshaw, R.Sidney and Y.L.Yin. *The RC6 Block Cipher. v1.1*. AES proposal, available on: `http://www.rsa.com/rsalabs/aes/`.
7. B.Schneier, J.Kelsey, D.Whiting, D.Wagner, C.Hall and N.Ferguson. *Twofish: a 128-bit block cipher*. AES proposal, available on: `http://www.counterpane.com/twofish.html`.
8. S.Vaudenay and S.Moriai. *Comparison of the randomness provided by some AES candidates*. Rump session at AES2.

Linear Cryptanalysis of Reduced-Round Versions of the SAFER Block Cipher Family

Jorge Nakahara Jr, Bart Preneel[⋆], and Joos Vandewalle

Katholieke Universiteit Leuven, Dept. Electrical Engineering–ESAT
Kardinaal Mercierlaan 94, B–3001 Heverlee, Belgium
{Jorge.Nakahara,Bart.Preneel,Joos.Vandewalle}@esat.kuleuven.ac.be

Abstract. This paper presents a linear cryptanalytic attack against reduced round variants of the SAFER family of block ciphers. Compared with the 1.5 round linear relations by Harpes *et al.*, the following new linear relations were found: a 3.75-round non-homomorphic linear relation for both SAFER-K and SAFER-SK with bias $\epsilon = 2^{-29}$; a 2.75 round relation for SAFER+ with bias $\epsilon = 2^{-49}$. For a 32-bit block mini-version of SAFER a 4.75-round relation with bias $\epsilon = 2^{-16}$ has been identified. These linear relations apply only to certain weak key classes. The results show that by considering non-homomorphic linear relations, more rounds of the SAFER block cipher family can be attacked. The new attacks pose no threat to any member of the SAFER family.

1 Introduction

SAFER (Secure And Fast Encryption Routine) is a family of block ciphers, designed by Massey, which comprises 64-bit block ciphers like SAFER-K64 [11], SAFER-K128 [12], SAFER-SK40, SAFER-SK64 and SAFER-SK128 [13]. The number that follows each cipher name indicates the key size. The newest member of this family is the AES candidate SAFER+ [10] designed jointly with Khachatrian and Kuregian; SAFER+ has a 128-bit block size and variable key size versions of 128, 192 and 256 bits. We will also analyze a 32-bit block mini-version, called SAFER-K32.

The more widespread, easy-to-deploy and better-understood an encryption algorithm is, the more attractive it becomes as a target for cryptanalysts. All SAFER family members have publicly available descriptions, are unpatented, royalty-free, with plenty of flexibility for different key sizes and block sizes, and are designed to be efficiently implementable in software [13]. These are key features to make SAFER+ widely deployed. An example is the inclusion of SAFER+ for authentication purposes in Bluetooth [1, p. 149]; this is the codename for a technology specification for low-cost, short range radio links between mobile PC's, mobile phones and other portable devices.

[⋆] F.W.O. research associate, sponsored by the Fund for Scientific Research, Flanders (Belgium).

Several theoretical attacks have been published on the ciphers of the SAFER family (in most cases versions were considered with a reduced number of rounds): differential cryptanalysis by Massey [12], truncated differentials by Knudsen and Berson [17], later improved by Wu *et al.* [7], an algebraic attack by Murphy [20], key schedule attacks by Knudsen [15] and by Kelsey *et al.* [9], and observations on the PHT design by Vaudenay [21] and Brincat *et al.* [2]. Linear cryptanalysis has been considered by Harpes *et al.* in [4] (see also [3]); they show that a generalized linear attack becomes infeasible for three or more rounds of SAFER-K64. This paper proposes an improved linear analysis by considering a wider class of linear relations; it also identifies certain classes of keys that are 'weak' w.r.t. linear cryptanalysis.

This paper is organized as follows. Section 2 describes the structure of SAFER-K64 and its key-schedule algorithm. Section 3 describes a 32-bit-block mini-version of SAFER-K. Section 4 introduces principles of linear cryptanalysis and some terminology for our attack. Section 5 gives particular features of a new type of linear relation for SAFER ciphers. Section 6 contains our results for the SAFER cipher family; their further use in an attack is described in Sect. 7. Section 8 discusses the methodology used to obtain the new linear relations and Sect. 9 summarizes the analysis results. Annex A presents a ciphertext-only attack.

2 Description of SAFER-K64

SAFER-K64 is a 64-bit-block iterated cipher with $r = 6$ rounds and a 64-bit user-selected key K. The key K is expanded into $2r + 1$ subkeys, that is, two subkeys per round plus one subkey for an output transformation. The following description of the round structure of SAFER-K64 also applies to SAFER-SK40, SAFER-SK64 and SAFER-SK128, because their ciphers only differ in the key schedule. Therefore, SAFER-K/-SK will be used as a notation when the analysis applies to both ciphers.

2.1 The Round Structure

In each encryption round, the input block B is first split into 8 bytes: $B = (b_1, b_2, b_3, b_4,\ b_5,\ b_6, b_7, b_8)$, $b_i \in \mathbb{Z}_{256}, 1 \leq i \leq 8$. Each byte b_j is combined with the first round-subkey K_{2i}: $Y = B + K_{2i} = (b_1 \oplus K_{2i}^1, b_2 \boxplus K_{2i}^2, b_3 \boxplus K_{2i}^3, b_4 \oplus K_{2i}^4, b_5 \oplus K_{2i}^5,\ b_6 \boxplus K_{2i}^6,\ b_7 \boxplus K_{2i}^7,\ b_8 \oplus K_{2i}^8)$ where $\oplus$ denotes bitwise XOR and $\boxplus$ represents ADD(ITION) modulo 256. Each byte of $Y = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$ is input to an S-box: $Z = (\mathrm{X}(y_1), \mathrm{L}(y_2), \mathrm{L}(y_3), \mathrm{X}(y_4), \mathrm{X}(y_5),\ \mathrm{L}(y_6),\ \mathrm{L}(y_7),\ \mathrm{X}(y_8))$, where X(.) is an eXponentiation S-box and L(.) a Logarithm S-box, described later. This S-box layer will be referred to as the non-linear or NL layer. Subsequently, $Z = (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8)$ is combined with the second round-subkey K_{2i+1}: $T = Z + K_{2i+1} = (z_1 \boxplus K_{2i+1}^1, z_2 \oplus K_{2i+1}^2, z_3 \oplus K_{2i+1}^3, z_4 \boxplus K_{2i+1}^4, z_5 \boxplus K_{2i+1}^5, z_6 \oplus K_{2i+1}^6, z_7 \oplus K_{2i+1}^7, z_8 \boxplus K_{2i+1}^8)$. Finally, the bytes of T are input to a linear transformation called Pseudo-Hadamard Transform or PHT layer.

The alternating XOR/ADD layer of input data with the first subkey bytes, together with the NL layer will be referred to as the NL half-round; similarly, the alternating ADD/XOR layer of intermediate data with the second subkey, together with the PHT layer will be called the PHT half-round.

There are two S-boxes: an eXponentiation $X(a) = (45^a \bmod 257) \bmod 256$ (X-box, for short), and a Logarithm $L(a) = \log_{45}(a) \bmod 257$ (or L-box, for short) for $a \neq 0$, with the special case $L(0) = 128$. They are each other's inverses, that is, $X(L(a)) = L(X(a)) = a, \forall a \in \mathbb{Z}_{256}$.

The PHT layer denotes a network of twelve 2-PHT boxes, where the latter is defined as $\text{2-PHT}(a,b) = (2 \cdot a \boxplus b, a \boxplus b)$, for $a, b \in \mathbb{Z}_{256}$. Denoting the input to a PHT layer by $Y = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$ and its output by $Z = (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8)$, where $y_i, z_i \in \mathbb{Z}_{256}$, $1 \leq i \leq 8$, this transformation can be described by $Z = Y^T \cdot M$, where M is called the PHT matrix:

$$M = \begin{pmatrix} 8 & 4 & 4 & 2 & 4 & 2 & 2 & 1 \\ 4 & 2 & 4 & 2 & 2 & 1 & 2 & 1 \\ 4 & 2 & 2 & 1 & 4 & 2 & 2 & 1 \\ 2 & 1 & 2 & 1 & 2 & 1 & 2 & 1 \\ 4 & 4 & 2 & 2 & 2 & 2 & 1 & 1 \\ 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 \\ 2 & 2 & 1 & 1 & 2 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} .$$

Let $T = (t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8)$ be the output after r rounds. There is an output transformation which mixes T with the last subkey, giving the ciphertext: $C = T + K_{2r+1} = (t_1 \oplus K_{2r+1}^1, t_2 \boxplus K_{2r+1}^2, t_3 \boxplus K_{2r+1}^3, t_4 \oplus K_{2r+1}^4, t_5 \oplus K_{2r+1}^5, t_6 \boxplus K_{2r+1}^6, t_7 \boxplus K_{2r+1}^7, t_8 \oplus K_{2r+1}^8)$. Decryption involves the application of the inverse of each round with reverse order for the subkeys. More details can be found in [11,12].

The round structure of SAFER+ uses the same S-boxes and 2-PHT primitives found in 64-bit block members, but the former uses a different PHT layer composed of four 2-PHT layers, and a particular fixed permutation between 2-PHT layers, called Armenian Shuffle (see Fig. 1).

2.2 The Key Schedule

The key schedule of SAFER-K64 accepts a 64-bit user-selected key K and generates 64-bit subkeys $K_i, 1 \leq i \leq 2r+1$, that is, two subkeys per round plus one subkey for the output transformation. K itself is used (unchanged) as the first subkey K_1. Subsequently, K is split into eight bytes, $(K^1, K^2, K^3, K^4, K^5, K^6, K^7, K^8)$, and each byte is left rotated by three bits. Next, fixed byte values called key bias $B_2^1, \ldots, B_2^8$ are added to bytes $K^1, \ldots, K^8$ respectively, where

$$B_i^j = (45^{45^{9i+j} \bmod 257} \bmod 257) \bmod 256 \ , \ 2 \leq i \leq 2r+1 \ , \ 1 \leq j \leq 8 \ .$$

The result is the second subkey $K_2 = (ROL_3(K^1) \boxplus B_2^1, \ldots, ROL_3(K^8) \boxplus B_2^8)$. The other subkeys are generated by following the same steps using the previous subkey as input: rotate each input byte left by 3 bits and add the next key bias.

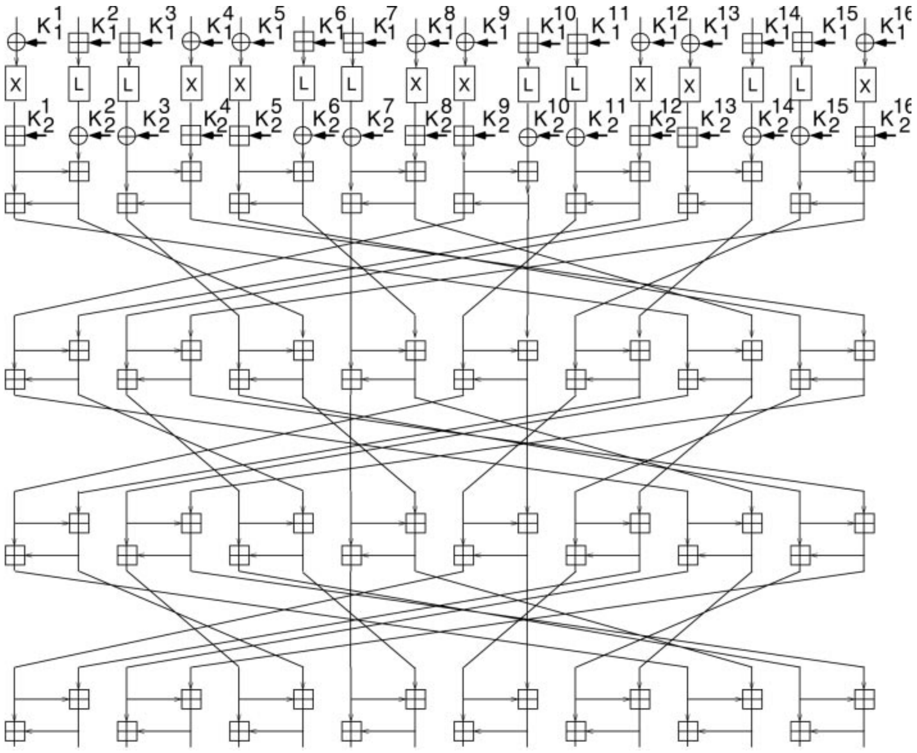


Fig. 1. One round of SAFER+

Key schedule weaknesses in SAFER-K64 were demonstrated by Knudsen [15], resulting in the improved key schedule of SAFER-SK64 [12]. Kelsey *et al.* have pointed out a weakness in the key schedule of SAFER+ for long keys [9]. Key schedule weaknesses will not be considered in this paper, but our analysis will point out that some keys are weak w.r.t. linear cryptanalysis.

3 A Mini-Version of SAFER-K64

Some block ciphers allows all of their individual components to be reduced to a half, a quarter or even smaller sizes, while the security level relative to the block size remains similar. This is also the case for the SAFER cipher family. This paper analyzes one such reduced version which will be called SAFER-K32. This is a 32-bit block cipher with a 32-bit user key, $r = 8$ rounds, and with S-boxes defined as $X(a) = (g^a \bmod 17) \bmod 16$, and $L(a) = \log_g a \mod 17$, for $a \neq 0$ and $L(0) = 8$. There are eight degrees of freedom in choosing g such that $GF(17) =< g >$, namely $g \in \{3, 5, 6, 7, 10, 11, 13, 14\}$ (see [17]). The value $g = 11$ was chosen arbitrarily for this mini-version.

The emphasis of the current analysis is not to attack the key schedule but the cipher itself; therefore it will be assumed that the key schedule for SAFER-K32 has a structure similar to that of SAFER-K64. The scale is reduced: the key schedule generates $(2r+1)$ 32-bit subkeys and it uses the same generator as the cipher.

The main reasons to consider reduced versions of ciphers are:

– the reduced dimensions allow a more comprehensive (exhaustive) analysis, to be carried out which is not always possible in the original cipher;
– it is hoped that weaknesses found in the mini-version can be extended to the larger cipher, or at least that they may provide some insight in potential weaknesses in the original cipher.

4 Linear Cryptanalysis of SAFER

4.1 Linear Cryptanalysis

Linear cryptanalysis is a statistical, known-plaintext attack introduced by Matsui and Yamagishi in 1992 in an attack against FEAL [19]. It was extended to DES in 1993 [18]. The attack explores (approximate) linear relations between plaintext, ciphertext and subkey bits. Linear approximations for an iterated cipher are usually made by combining approximations for each round.

If $X_i = (x_n, x_{n-1}, \ldots, x_2, x_1)$ is an n-bit input to a round, $R(X_i)$ is its output, and K_i the round subkey, then a linear relation can be expressed as

$$X_i \cdot \Gamma I \oplus R(X_i) \cdot \Gamma O = K_i \cdot \Gamma K_i \ , \tag{1}$$

where $\Gamma I, \Gamma O$ and ΓK_i are n-bit masks which specify the bits of X_i, $R(X_i)$ and K_i involved in the linear relation. For example, $X_i \cdot \Gamma I = X \cdot 45_x = x_1 \oplus x_3 \oplus x_7$ (the subscript 'x' indicates hexadecimal values).

The left-hand side of equation (1) provides an estimate for the xor of the subkey bits on the right-hand side. Without loss of generality, the following simplified equation is employed

$$X_i \cdot \Gamma I \oplus R(X_i) \cdot \Gamma O = 0 \ . \tag{2}$$

Two numerical values can be associated with (2). First, a probability $p = \Pr(X_i \cdot \Gamma I = R(X_i) \cdot \Gamma O)/2^n$ that expresses the frequency with which equation (2) holds (relation (2) is also called a linear approximation). Second, the deviation of parity of (2) from a random relation, or $p' = p - \frac{1}{2}$. It is clear that $-\frac{1}{2} \leq p' \leq \frac{1}{2}$ and the approximation is useful only if $p' \neq 0$. The absolute value $\epsilon = |p'|$ is called bias [8]. The larger the bias the more useful the linear relation is, that is, the more unbalanced the parity of (2) from a random distribution the less plaintext is needed to estimate the value of $K_i \cdot \Gamma K_i$ (with high degree of assurance). The number N of known plaintexts required for an attack using a linear relation with bias ϵ equals $N = c \cdot \epsilon^{-2}$, where c is a small constant, which depends on the algorithm used for the estimation [8,18]. In case $p' < 0$, the value obtained for $K_i \cdot \Gamma K_i$ is actually $\overline{K_i \cdot \Gamma K_i} = (K_i \cdot \Gamma K_i) \oplus 1$.

The following notation will be used to represent a binary-valued linear relation for one round of an iterated (n-bit block) cipher:

$$\Gamma = (\Gamma I, \Gamma O, \epsilon) \ . \tag{3}$$

One-round linear relations can be concatenated or stacked in order to approximate more rounds. If $\Gamma_1 = (\Gamma X_1, \Gamma Y_1, \epsilon_1)$, $\Gamma_2 = (\Gamma X_2, \Gamma Y_2, \epsilon_2)$ are r_1-round and r_2-round independent linear relations, respectively and $\Gamma Y_1 = \Gamma X_2$, then it is possible to combine them to form an $(r_1 + r_2)$-round linear relation $\Gamma_3 = (\Gamma X_1, \Gamma Y_2, \epsilon)$ with bias $\epsilon = 2 \cdot \epsilon_1 \cdot \epsilon_2$ (Matsui's Piling-Up lemma [18]). Note however that this assumes that the subkeys are mutually independent and uniformly distributed which is not the case for any member of the SAFER cipher family, when the key schedule algorithms are used. Nonetheless, practical experiments show that the subkeys generated through the respective key schedules of each cipher are adequately randomized in order for the approximations to hold.

As an example of linked relation, a one-round linear relation for SAFER-K64 can be viewed as the concatenation of two half-round linear relations: $\Gamma_{NL} = (\Gamma X, \Gamma M, \epsilon_1)$, and $\Gamma_{PHT} = (\Gamma M, \Gamma Y, \epsilon_2)$, where ΓM denotes a bit-mask applied to the intermediate value in the middle of a round, between the output of the NL and the input to the PHT layers.

4.2 Homomorphic Linear Relations

Definition 1. *Let G_1 and G_2 be groups with operations $\otimes$ and $\boxdot$, respectively. A mapping M from G_1 into G_2 is called a* homomorphism *if*

$$M(y \otimes z) = M(y) \boxdot M(z) \ , \ \forall y, z \in G_1 \ . \tag{4}$$

Definition 2. *A binary-valued function f is* balanced *if it outputs the value 0 for exactly half of its inputs.*

Definition 3 (Harpes-Kramer-Massey [3,4]). *An* I/O sum *$S^{(i)}$ for a round is a modulo-two sum of a balanced binary-valued function f_i of the round input $Y^{(i-1)}$ and a balanced binary-valued function g_i of the round output $Y^{(i)}$, namely*

$$S^{(i)} = f_i(Y^{(i-1)}) \oplus g_i(Y^{(i)}) \ . \tag{5}$$

The functions f_i and g_i are called input function and output function, respectively, of the I/O sum $S^{(i)}$. I/O sums for successive rounds will be called linked if the output function of each I/O sum except the last coincides with the input function of the following I/O sum: $g_i = f_{i+1}$. When $S^{(1)}, S^{(2)}, \ldots, S^{(r)}$ are linked, then their sum is also an I/O sum:

$$S^{(1\ldots r)} = \bigoplus_{i=1}^{r} S^{(i)} = f_0(Y^{(0)}) \oplus g_r(Y^{(r)}) \ . \tag{6}$$

which will be called an r-round I/O sum.

Harpes *et al.* report in [3,4] that SAFER-K64 is immune to a generalization of linear cryptanalysis [18] which involves only homomorphic I/O sums after 1.5 rounds. Namely, the best homomorphic I/O sum is stated as the concatenation of the following NL-PHT-NL half-rounds:

$$\begin{array}{llll} (\mathtt{000000zz000000zz_x}, & \mathtt{0000000100000001_x}, & 2\cdot(\frac{28}{256})^2) & \text{(NL half-round)} \\ (\mathtt{0000000100000001_x}, & \mathtt{0001000000000000_x}, & 2^{-1}) & \text{(PHT half-round)} \\ (\mathtt{0001000000000000_x}, & \mathtt{00zz000000000000_x}, & \frac{28}{256}) & \text{(NL half-round)}\ , \end{array} \tag{7}$$

where $\mathtt{zz} \in \{\mathtt{cd_x}, \mathtt{ff_x}\}$, and the overall bias (using Matsui's Piling Up Lemma) is $\epsilon = 2^2 \cdot (\frac{28}{256})^3 \approx 2^{-7.58}$. Using homomorphic linear relations, the bit-masks which are used to approximate one round of SAFER-K64 take into account the group operations used to mix subkey bits in each round. In this way the effect of carry bits is avoided when the group operation is addition modulo 256.

The homomorphicity of the masking function is important for the approximation of subkey bits mixed in a round, or more specifically, for the group operations used to mix subkey bits in a round. For example, let $G_1 = (\mathbb{Z}_{256}, \boxplus)$ and $G_2 = (\mathbb{Z}_{256}, \oplus)$ be groups. The only homomorphic masking function in this setting is $M_1(y) = \Gamma \cdot y = \mathtt{01_x} \cdot y$, that is, the mask which takes only the least significant bit (LSB). In [3] it is stated that (7) is the best homomorphic linear relation achievable for SAFER-K/-SK. In the next sections, it will be shown that approximations using non-homomorphic bit-masks result in improved linear relations.

4.3 Non-homomorphic Linear Relations

Let $M_2(y) = \mathtt{02_x} \cdot y$ be a masking function and K be a subkey byte. The M_2 mapping is non-homomorphic, because

$$M_2(y \boxplus K) = \mathtt{02_x} \cdot (y \boxplus K) \neq M_2(y) \oplus M_2(K) = \mathtt{02_x} \cdot y \oplus \mathtt{02_x} \cdot K\ .$$

This happens because of a possible carry bit that can propagate from the LSB to the second LSB. Assuming that the intermediate data values in a round and the subkey bits are uniformly distributed, this carry bit only exists with probability 1/4. Therefore, a bias penalty of 2^{-2} is to be accounted for.

If one considers the bit-masks for the subkey bytes applied only to a fraction of a round, like a subkey-mixing layer, then one can split a one round approximation into quarters of a round. Therefore, the bit-masks that make up the approximation only of the mixing layer of subkey bits in a round, or only of the NL layer, or only of the PHT layer will be called quarter-round approximations. As an example, $(\mathtt{0002020100000201_x}, \mathtt{0001010200000102_x}, 2^{-16})$ is an NL quarter-round approximation of the S-box layer in a round of SAFER-K64. Such partial approximations will be used later for (fractional) linear attacks.

5 Key-Dependent Linear Relations

The non-homomorphicity of some bit-masks has two important consequences for the corresponding linear relations:

(1) Let $X, S, K_i \in \mathbb{Z}_{256}$ be the input, the output and the subkey bytes in a subkey addition operation in a round of SAFER-K/-SK, that is, $S = X \boxplus K_i$. An approximation for the addition operation, using bit-masks $\texttt{02}_\texttt{x}$, take the form $S \cdot \texttt{02}_\texttt{x} = X \cdot \texttt{02}_\texttt{x} \boxplus K_i \cdot \texttt{02}_\texttt{x}$. This approximation is non-homomorphic. Besides, it assumes that there is no carry bit into the second LSB position, otherwise the approximation would be void. This carry bit can be avoided if the least significant bit of subkey K_i is zero. Other non-homomorphic masks present similar dependencies on the subkey bits.

(2) If the carry bit restrictions are satisfied for the non-homomorphic bit-masks as in the item (1) above, then the masks become homomorphic. In this way, the overall bias of the corresponding linear relation is increased, as there is no more bias penalty to account for. Therefore, one cannot only control the carry propagation (assuring the approximations for the addition of subkey bytes) but also improve the overall bias. The restricted validity of the approximation to subkeys which possess a certain bit-pattern is only apparent. By specifying bit-masks for each key class according to the different approximations of the two LSBs in the addition, all keys in the key space can be attacked. For example, let $X = (x_{n-1}, \ldots, x_1, x_0)$ and K be the input and $S = (s_{n-1}, \ldots, s_1, s_0) = X \boxplus K$, and ΓK be the key bit-mask. Then, for the bit-masks which explore only the two LSBs of addition of subkey bytes, there are the following possibilities:

   (a) Let the approximation be $S \cdot \texttt{02}_\texttt{x} = X \cdot \texttt{02}_\texttt{x} \oplus K \cdot \Gamma K$ or $s_1 = x_1 \oplus K \cdot \Gamma K$. As the expression of addition is $s_1 = x_1 \oplus k_1 \oplus x_0 \cdot k_0$, it follows that $k_0 = 0$ and $\Gamma K = \texttt{02}_\texttt{x}$.

   (b) Let the approximation be $S \cdot \texttt{02}_\texttt{x} = X \cdot \texttt{03}_\texttt{x} \oplus K \cdot \Gamma K$ or $s_1 = x_1 \oplus x_0 \oplus K \cdot \Gamma K$. As the expression of addition is $s_1 = x_1 \oplus k_1 \oplus x_0 \cdot k_0$, it follows that $k_0 = 1$ and $\Gamma K = \texttt{02}_\texttt{x}$.

   (c) Let the approximation be $S \cdot \texttt{03}_\texttt{x} = X \cdot \texttt{02}_\texttt{x} \oplus K \cdot \Gamma K$ or $s_1 \oplus s_0 = x_1 \oplus K \cdot \Gamma K$. As the expression of addition is $s_1 \oplus s_0 = x_1 \oplus k_1 \oplus x_0 \cdot k_0 \oplus x_0 \oplus k_0$, it follows that $k_0 = 1$ and $\Gamma K = \texttt{03}_\texttt{x}$.

   (d) Let the approximation be $S \cdot \texttt{03}_\texttt{x} = X \cdot \texttt{03}_\texttt{x} \oplus K \cdot \Gamma K$ or $s_1 \oplus s_0 = x_1 \oplus x_0 \oplus K \cdot \Gamma K$. As the expression of addition is $s_1 \oplus s_0 = x_1 \oplus k_1 \oplus x_0 \cdot k_0 \oplus x_0 \oplus k_0$, it follows that $k_0 = 0$ and $\Gamma K = \texttt{03}_\texttt{x}$.

   Therefore, each bit-mask imposes a different restriction on the key bit pattern but also includes all possibilities for the LSB of the key. For the bit-masks in (a) and (d) the key bit k_0 might be 0, and for the bit-masks in (b) and (c), k_0 is required to be 1. Although the bit masks in the linear relations in the next section are valid for certain specific key bit patterns, they can easily be changed to cover each different key class in the key space.

6 Search Results

We now present the results of our search for non-homomorphic linear relations for the different members of the SAFER family.

Definition 4. *In a linear approximation, an S-box is said to be* active *if the approximation applies a non-zero output bit-mask to that S-box. The number of active S-boxes in a linear relation will be denoted with $\mathcal{S}$.*

A search for linear relations of SAFER-K/-SK resulted in a 3.75-round linear relation with $\mathcal{S} = 7$ and theoretical bias $\epsilon_1 = 2^{-39}$:

$$\begin{array}{llll}
(\texttt{0102010201020102}_x & , \texttt{0000000000020000}_x & , 2^{-5}) & \text{(PHT half-round)} \\
(\texttt{0000000000020000}_x & , \texttt{0100000001000000}_x & , 2^{-5}) & \text{(one round)} \\
(\texttt{0100000001000000}_x & , \texttt{0002000200020003}_x & , 2^{-11}) & \text{(one round)} \\
(\texttt{0002000200020003}_x & , \texttt{0002000100000000}_x & , 2^{-18}) & \text{(one round)} \\
(\texttt{0002000100000000}_x & , \texttt{0002000100000000}_x & , 2^{-2}) & \text{(subkey quarter-round)} \ .
\end{array} \tag{8}$$

Recalling the key dependency discussed in Sect. 5, item (1), the following restrictions on subkey bits are necessary for the approximation of subkey addition in a 1.25R attack on five rounds SAFER-K/-SK to hold:

$$\mathrm{LSB}(K_2^4, K_2^8, K_3^6, K_6^1, K_6^5, K_7^2, K_7^6, K_8^4, K_8^8, K_9^2) = 0 \tag{9}$$

where the notation $\mathrm{LSB}(\cdot, \ldots, \cdot) = 0$ means that the least significant bit of each argument is zero. Therefore, the actual bias of relation (8) is $\epsilon_1^* = 2^{-29}$. These keys are called weak keys w.r.t. relation (8). Incidentally, these ten key bits in (9) map to exactly ten different user key bits, according to the key schedule of SAFER-K64 [11] which means that one in 1024 user keys is weak. For the key schedule of SAFER-SK64 (see [13]), these ten key bits imply conditions on 16 different user key bits.

Recalling the discussion in Sect. 5, item (2), the bit-masks in (8) can be adapted accordingly to satisfy the other 1023 subkey classes. For example,

$$\begin{array}{llll}
(\texttt{0102010301020102}_x & , \texttt{0000000000020000}_x & , 2^{-5}) & \text{(PHT half-round)} \\
(\texttt{0000000000020000}_x & , \texttt{0100000001000000}_x & , 2^{-5}) & \text{(one round)} \\
(\texttt{0100000001000000}_x & , \texttt{0002000200020003}_x & , 2^{-11}) & \text{(one round)} \\
(\texttt{0002000200020003}_x & , \texttt{0002000100000000}_x & , 2^{-18}) & \text{(one round)} \\
(\texttt{0002000100000000}_x & , \texttt{0002000100000000}_x & , 2^{-2}) & \text{(subkey quarter-round)}
\end{array} \tag{10}$$

has the same theoretical bias as (8), but the weak key restrictions are:

$$\mathrm{LSB}(K_2^4, K_2^8, K_3^6, K_6^1, K_6^5, K_7^2, K_7^6, K_8^4, K_8^8, K_9^2) = 0 \quad , \tag{11}$$

which imply a different weak key class. The actual bias though, is the same as before, $\epsilon = 2^{-29}$. Similarly, changing each bit-mask in the addition of subkey bytes, in each round, one can get relations which hold for each key in the key space. The same observation holds for the linear relations of SAFER+ and SAFER-K32 below.

For SAFER+ the following linear relation with $\mathcal{S} = 12$ and 2.75 rounds was found:

$$\begin{aligned}
&(\texttt{000201020100000020002000202000100}_\texttt{x}\ ,\alpha\ ,2^{-11})\ \text{(PHT half-round)}\\
&(\alpha\ ,\beta\ ,2^{-29})\ \text{(one round)}\\
&(\beta\ ,\gamma\ ,2^{-27})\ \text{(one round)}\\
&(\gamma\ ,\gamma\ ,\ 2^{-5}\ \ \text{(subkey quarter round)}
\end{aligned} \tag{12}$$

with theoretical bias $\epsilon_3 = 2^{-69}$, $\alpha = \texttt{00000200000202000000202000020200}_\texttt{x}$, $\beta = \texttt{0000000101000000001000000001000001}_\texttt{x}$, $\gamma = \texttt{02000002000203010203020102000100}_\texttt{x}$. The following key bit conditions are necessary for the approximation of subkey addition in a 1.25R attack on four rounds of SAFER+ to hold:

$$\begin{aligned}
\text{LSB}(K_2^4, K_2^8, K_2^{12}, K_2^{13}, K_3^3, K_3^6, K_3^7, K_3^{10}, K_3^{11}, K_3^{14}) = 0\ ,\\
\text{LSB}(K_3^{15}, K_6^4, K_6^5, K_6^9, K_6^{13}, K_6^{16}, K_7^6, K_7^7, K_7^{10}, K_7^{11}) = 0\ .
\end{aligned} \tag{13}$$

Therefore, the actual bias of (12) is $\epsilon_3^* = 2^{-49}$.

Linear cryptanalysis of SAFER-K32 resulted in a 4.75 round linear relation with $\mathcal{S} = 9$:

$$\begin{aligned}
&(\texttt{12121212}_\texttt{x}\ ,\texttt{00000200}_\texttt{x}\ ,2^{-5})\ \text{(PHT half-round)}\\
&(\texttt{00000200}_\texttt{x}\ ,\texttt{10001000}_\texttt{x}\ ,2^{-3})\ \text{(one round)}\\
&(\texttt{10001000}_\texttt{x}\ ,\texttt{02020203}_\texttt{x}\ ,2^{-7})\ \text{(one round)}\\
&(\texttt{02020203}_\texttt{x}\ ,\texttt{02010000}_\texttt{x}\ ,2^{-8})\ \text{(one round)}\\
&(\texttt{02010000}_\texttt{x}\ ,\texttt{32110000}_\texttt{x}\ ,2^{-6})\ \text{(one round)}\\
&(\texttt{32110000}_\texttt{x}\ ,\texttt{32110000}_\texttt{x}\ ,2^{-2})\ \text{(subkey quarter-round)}
\end{aligned} \tag{14}$$

with theoretical bias $\epsilon_4 = 2^{-28}$. The following restrictions are needed for the approximation of subkey addition in a 1.25R attack on six rounds of SAFER-K32 to hold:

$$\text{LSB}(K_2^4, K_2^8, K_3^6, K_6^1, K_6^5, K_7^2, K_7^6, K_8^4, K_8^8, K_9^2, K_{10}^4, K_{11}^2) = 0 \tag{15}$$

and the actual bias of (14) is $\epsilon_4^* = 2^{-16}$.

7 Fractional Linear Attacks

In Sect. 4.3 linear approximations were described that covered only part of a SAFER-K/-SK round, for example, a half- or a quarter-round. Linear attacks using such fractional linear relations include fractions of a round only at the beginning and end of the cipher, and will be denoted fractional attacks. In the following the subscript x will sometimes be omitted from bit masks to simplify notation, for example, 02 instead of 02_x; the interpretation should be clear from the context.

As an example of fractional attack, relation (8) can be used in a 1.25R attack. This is an analogy with the usual 1R or 2R attacks which only discard full rounds [6,18]. This 1.25R attack does not include the first half of the first round neither the last three quarters of the last round in the approximation, that is, the linear relation (8) does not cover 1.25 rounds (see Fig. 2). The idea for our attacks is to place the linear relations between two subkey layers, such that subkeys at both ends of the cipher are identified. A similar description applies for other fractional values.

This attack covers five rounds of SAFER-K/-SK, without the output transformation, and identifies 81 subkey bits as follows, assuming the weak-key conditions (9) are satisfied:

– let $P_1, \ldots, P_8$ be plaintext bytes, $D_1, \ldots, D_8$ be the result of applying the inverse of the PHT layer (which is unkeyed) to the ciphertext bytes, and $X(.)$ and $L(.)$ the S-boxes. The following linear relation, derived from (8), can be used:

$$\begin{aligned} &X(P_1 \oplus K_1^1) \cdot 01 \oplus L(P_2 \boxplus K_1^2) \cdot 02 \oplus L(P_3 \boxplus K_1^3) \cdot 01 \;\oplus \\ &X(P_4 \oplus K_1^4) \cdot 02 \oplus X(P_5 \oplus K_1^5) \cdot 01 \oplus L(P_6 \boxplus K_1^6) \cdot 02 \;\oplus \\ &L(P_7 \boxplus K_1^7) \cdot 01 \oplus X(P_8 \oplus K_1^8) \cdot 02 \oplus X(D_2 \boxplus K_8^2) \cdot 02 \;\oplus \\ &\qquad L(D_4 \boxminus K_8^4) \cdot 01 = K_i \cdot \Gamma K_i \quad , \end{aligned} \tag{16}$$

where $\boxminus$ denotes subtraction in $\mathbb{Z}_{256}$.

– the $K_i \cdot \Gamma K_i$ bit is the following: $(K_2^2 \oplus K_2^4 \oplus K_2^6 \oplus K_2^8 \oplus K_3^6 \oplus K_6^1 \oplus K_6^5 \oplus K_7^2 \oplus K_7^4 \oplus K_8^4 \oplus K_8^8 \oplus K_9^2) \cdot 02 \oplus (K_2^1 \oplus K_2^3 \oplus K_2^5 \oplus K_2^7 \oplus K_4^6 \oplus K_5^1 \oplus K_5^5 \oplus K_8^2 \oplus K_8^6 \oplus K_9^4) \cdot 01 \oplus K_7^8 \cdot 03$, and

– the other 80 subkey bits are $K_1^1 \cdot \texttt{ff}$, $K_1^2 \cdot \texttt{ff}$, $K_1^3 \cdot \texttt{ff}$, $K_1^4 \cdot \texttt{ff}$, $K_1^5 \cdot \texttt{ff}$, $K_1^6 \cdot \texttt{ff}$, $K_1^7 \cdot \texttt{ff}$, $K_1^8 \cdot \texttt{ff}$, $K_9^2 \cdot \texttt{ff}$, $K_9^4 \cdot \texttt{ff}$. They can be identified with about $N \approx (2^{-29})^{-2} = 2^{58}$ known plaintext blocks using the maximum likelihood methods from [18].

Similarly, (12) can be used in a 1.25R-attack on four rounds of SAFER+, assuming weak key conditions (13) hold, as follows:

– let $P_1, \ldots, P_{16}$ be plaintext bytes, $D_1, \ldots, D_{16}$ be the result of applying the inverse of the PHT layer (which is unkeyed) to the ciphertext bytes, and $X(.)$ and $L(.)$ the S-boxes. The linear relation has the form:

$$\begin{aligned} &L(P_2 \boxplus K_1^2) \cdot 02 \oplus L(P_3 \boxplus K_1^3) \cdot 01 \oplus X(P_4 \oplus K_1^4) \cdot 02 \;\oplus \\ &X(P_5 \oplus K_1^5) \cdot 01 \oplus X(P_8 \oplus K_1^8) \cdot 02 \oplus L(P_{10} \boxplus K_1^{10}) \cdot 02 \;\oplus \\ &X(P_{12} \oplus K_1^{12}) \cdot 02 \oplus X(P_{13} \oplus K_1^{13}) \cdot 02 \oplus L(P_{15} \boxplus K_1^{15}) \cdot 01 \;\oplus \\ &L(D_1 \boxminus K_8^1) \cdot 02 \oplus L(D_4 \boxminus K_8^4) \cdot 02 \oplus X(D_6 \oplus K_8^6) \cdot 02 \;\oplus \\ &X(D_7 \oplus K_8^7) \cdot 03 \oplus L(D_8 \boxminus K_8^8) \cdot 01 \oplus L(D_9 \boxminus K_8^9) \cdot 02 \;\oplus \\ &X(D_{10} \oplus K_8^{10}) \cdot 03 \oplus X(D_{11} \oplus K_8^{11}) \cdot 02 \oplus L(D_{12} \boxminus K_8^{12}) \cdot 01 \;\oplus \\ &L(D_{13} \boxminus K_8^{13}) \cdot 02 \oplus X(D_{15} \oplus K_8^{15}) \cdot 01 = K_i \cdot \Gamma K_i \; . \end{aligned} \tag{17}$$

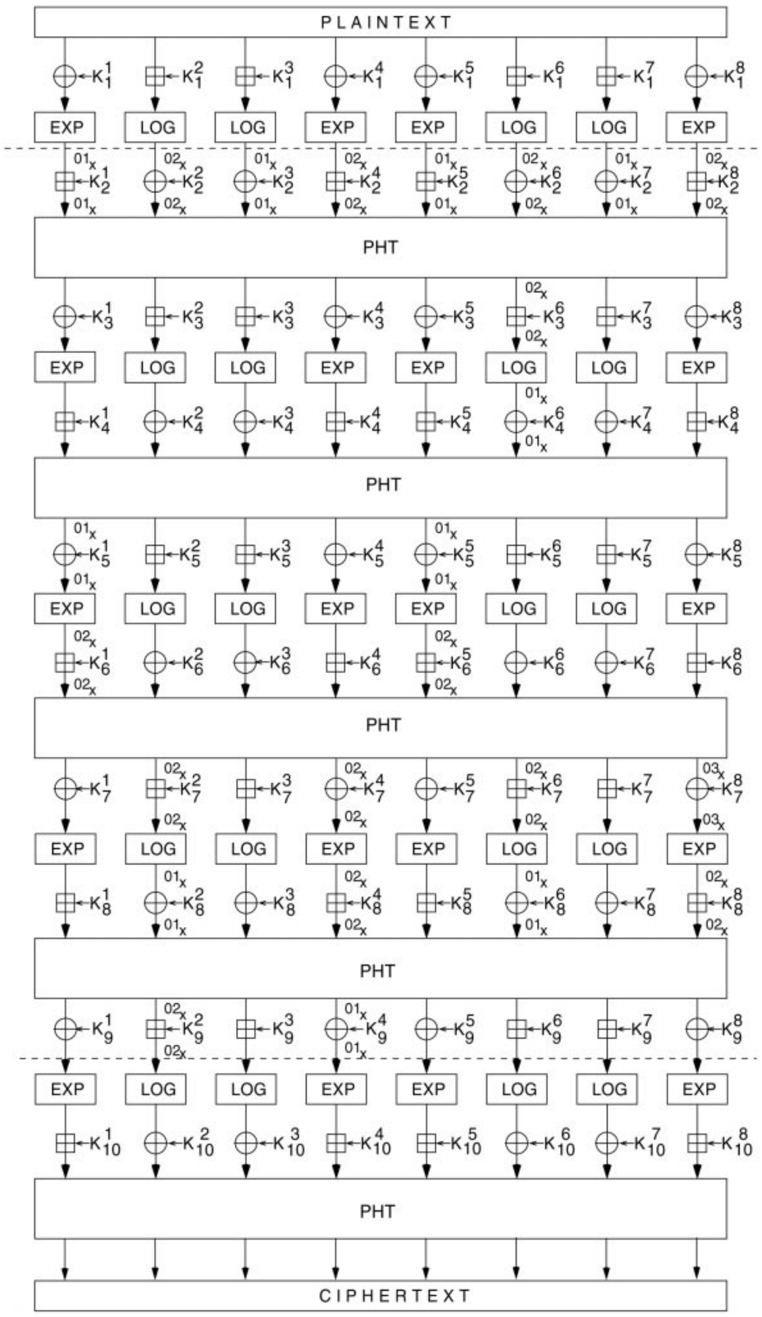


Fig. 2. A 1.25R attack on five rounds of SAFER-K/-SK (only non-zero bit-masks are shown)

- the $K_i \cdot \Gamma K_i$ bit is: $(K_2^2 \oplus K_2^4 \oplus K_2^8 \oplus K_2^{10} \oplus K_2^{12} \oplus K_2^{13} \oplus K_3^3 \oplus K_3^6 \oplus K_3^7 \oplus K_3^{10} \oplus K_3^{11} \oplus K_3^{14} \oplus K_3^{15} \oplus K_6^4 \oplus K_6^5 \oplus K_6^9 \oplus K_6^{13} \oplus K_6^{13} \oplus K_6^{16} \oplus K_7^1 \oplus K_7^4 \oplus K_7^6 \oplus K_7^9 \oplus K_7^{11} \oplus K_7^{13}) \cdot \mathtt{02} \oplus (K_7^7 \oplus K_7^{10}) \cdot \mathtt{03} \oplus (K_2^3 \oplus K_2^5 \oplus K_2^{15} \oplus K_4^3 \oplus K_4^6 \oplus K_4^7 \oplus K_4^{10} \oplus K_4^{11} \oplus K_4^{14} \oplus K_4^{15} \oplus K_5^4 \oplus K_5^5 \oplus K_5^9 \oplus K_5^{13} \oplus K_5^{16} \oplus K_7^8 \oplus K_7^{12} \oplus k_7^{15}) \cdot \mathtt{01}$ and
- the other 160 subkey bits are: $K_1^2 \cdot \mathtt{ff}$, $K_1^3 \cdot \mathtt{ff}$, $K_1^4 \cdot \mathtt{ff}$, $K_1^5 \cdot \mathtt{ff}$, $K_1^8 \cdot \mathtt{ff}$, $K_1^{10} \cdot \mathtt{ff}$, $K_1^{12} \cdot \mathtt{ff}$, $K_1^{13} \cdot \mathtt{ff}$, $K_1^{15} \cdot \mathtt{ff}$, $K_8^1 \cdot \mathtt{ff}$, $K_8^4 \cdot \mathtt{ff}$, $K_8^6 \cdot \mathtt{ff}$, $K_8^7 \cdot \mathtt{ff}$, $K_8^8 \cdot \mathtt{ff}$, $K_8^9 \cdot \mathtt{ff}$, $K_8^{10} \cdot \mathtt{ff}$, $K_8^{11} \cdot \mathtt{ff}$, $K_8^{12} \cdot \mathtt{ff}$, $K_8^{13} \cdot \mathtt{ff}$, $K_8^{15} \cdot \mathtt{ff}$. They can be identified using maximum likelihood techniques with about $N \approx (2^{-49})^{-2} = 2^{98}$ known plaintext blocks.

Finally, (14) leads to the following 1.25R attack on six rounds of SAFER-K32 (using weak keys) without the output transformation:

- let $P_1, \ldots, P_8$ be plaintext nibbles (4 bits), $D_1, \ldots, D_8$ be the result of applying the inverse of the PHT layer (which is unkeyed) to the ciphertext nibbles, and $X(.)$ and $L(.)$ the S-boxes. The linear relation has the form:

$$\begin{aligned} X(P_1 \oplus K_1^1) \cdot \mathtt{1} \oplus L(P_2 \boxplus K_1^2) \cdot \mathtt{2} \oplus L(P_3 \boxplus K_1^3) \cdot \mathtt{1} \ \oplus \\ X(P_4 \oplus K_1^4) \cdot \mathtt{2} \oplus X(P_5 \oplus K_1^5) \cdot \mathtt{1} \oplus L(P_6 \boxplus K_1^6) \cdot \mathtt{2} \ \oplus \\ L(P_7 \boxplus K_1^7) \cdot \mathtt{1} \oplus X(P_8 \oplus K_1^8) \cdot \mathtt{2} \oplus L(D_1 \boxplus K_8^1) \cdot \mathtt{3} \ \oplus \\ X(D_2 \oplus K_8^2) \cdot \mathtt{2} \oplus X(D_3 \oplus K_8^3) \cdot \mathtt{1} \oplus L(D_4 \boxminus K_8^4) \cdot \mathtt{1} = K_i \cdot \Gamma K_i \ . \end{aligned} \tag{18}$$

- the $K_i \cdot \Gamma K_i$ bit is: $(K_2^1 \oplus K_2^3 \oplus K_2^5 \oplus K_2^7 \oplus K_4^6 \oplus K_5^1 \oplus K_5^5 \oplus K_8^2 \oplus K_8^6 \oplus K_9^4 \oplus K_{10}^2 \oplus K_{11}^3 \oplus K_{11}^4) \cdot \mathtt{1} \oplus (K_7^8 \oplus K_{11}^1) \cdot \mathtt{3} \oplus (K_2^2 \oplus K_2^4 \oplus K_2^6 \oplus K_2^8 \oplus K_3^6 \oplus K_6^1 \oplus K_6^5 \oplus K_7^2 \oplus K_7^4 \oplus K_7^6 \oplus K_8^4 \oplus K_8^8 \oplus K_9^2 \oplus K_{10}^4 \oplus K_{11}^2) \cdot \mathtt{2}$ and then
- the other 48 subkey bits to be found are: $K_1^1 \cdot \mathtt{f}$, $K_1^2 \cdot \mathtt{f}$, $K_1^3 \cdot \mathtt{f}$, $K_1^4 \cdot \mathtt{f}$, $K_1^5 \cdot \mathtt{f}$, $K_1^6 \cdot \mathtt{f}$, $K_1^7 \cdot \mathtt{f}$, $K_1^8 \cdot \mathtt{f}$, $K_{12}^1 \cdot \mathtt{f}$, $K_{12}^2 \cdot \mathtt{f}$, $K_{12}^3 \cdot \mathtt{f}$, $K_{12}^4 \cdot \mathtt{f}$ using maximum likelihood methods with $N \approx (2^{-16})^{-2} = 2^{32}$ known plaintext blocks.

8 Methodology

The following procedure was used in order to obtain relations (8), (12) and (14):

- Initially, a Linear Approximation Table (LAT) for the S-boxes X and L was generated, containing for all possible input and output bit-masks the corresponding deviation value. Denoting by ΓI and ΓO general input and output bit-masks, and by $S(.)$ an S-box, each entry in the LAT contains

$$\mathrm{LAT}[\Gamma I, \Gamma O] = \Pr(I \cdot \Gamma I = S(I) \cdot \Gamma O) - \frac{1}{2} \tag{19}$$

  for all possible inputs $I \in \mathbb{Z}_{256}$. Only one table is actually needed, because the X-box is the inverse of the L-box, and for the latter one can swap the input and output masks to obtain the corresponding linear approximations.

- The approximation of subkey addition and xor was made separately and then evaluated together with the approximations for the S-boxes, confirming the key dependency (for addition). Indeed, the bias decreases if the subkeys do not exhibit a pattern that allows the expected bit-mask approximation. For example, the mask $M_2(x) = \texttt{02}_\texttt{x} \cdot x$, in the output of an addition operation with odd-valued key bytes gives zero bias; otherwise, the overall bias is the one provided by the xor of subkey and the X-box approximations.
- The approach taken for generating linear relations for the PHT layer was not exhaustive as was done for the S-boxes. Due to the addition operation performed in the 2-PHT boxes, it was observed that the most biased linear relations through the PHT layers would explore preferably the LSB(s) of the 2-PHT because they are least affected by carry bits. Besides, exploring few LSBs would require less weak-key restrictions. It was decided, arbitrarily, to concentrate efforts in the two LSBs only of each 2-PHT. A linear hull approximation was made for the PHT layer, because it was observed that linking together local (non-zero bias) approximations for the 2-PHT boxes could sometimes result in linear approximations for the PHT layer with zero bias. Indeed, some component relations of the linear hull have positive and others negative deviation with the same absolute value, which can cancel the effect of each other.
  Note that a (basic) linear relation tracks a single (approximation) path between input and output bits of a round component. A linear hull [14] corresponds to a set of linear relations all of which share the same input and output bit-masks, but each relation takes different paths across the component.
- The next step consisted in combining linear hulls for the (NL+subkey) layers with others for the PHT layer, in order to generate one-round linear approximations (hulls). Further, these one-round relations were combined either on top or at the bottom end of each other, in order to get as long a linear relation as possible. Such stacking strategy of combining one-round approximations was based on the idea of the inside-out attack of Wagner [5]. While constructing the final linear hull an important restriction was to try to keep the number of active S-boxes as small as possible from one round to the next, both in order to control the overall bias as well as to avoid attacking too many subkey bits at both ends of the cipher. For a linear relation of bias ϵ, the known-plaintext requirements for an effective (high success rate) linear attack on $2n$-bit block ciphers is $N \geq (\epsilon)^{-2}$, that is, $\epsilon \geq (\sqrt{2^{2n}})^{-1} = 2^{-n}$. Therefore, an immediate restriction for 64-bit-block cipher versions is $\epsilon \geq 2^{-32}$. Similarly, for SAFER+, $\epsilon \geq 2^{-64}$, and for SAFER-K32, $\epsilon \geq 2^{-16}$.

9 Conclusion

In this paper, SAFER-K32, SAFER-K/-SK and SAFER+ ciphers were analyzed using non-homomorphic linear cryptanalysis. Table 1 summarizes our results.

The algorithm used for our attack uses Matsui's idea of keeping only the highest parity counter(s) (say, for the best ten key candidates). The advantage

Table 1. Linear relations found for the SAFER cipher family

	Cipher		
	SAFER-K32	SAFER-K/-SK	SAFER+
# rounds lin. rel.	4.75	3.75	2.75
Bias (weak keys)	2^{-16}	2^{-29}	2^{-49}
Attack type	1.25R	1.25R	1.25R
# subkey bits	48+1 †	80+1 †	160+1 †
Time complexity (parity computation)	$2^{48+2\cdot 16}$	$2^{80+2\cdot 29}$	$2^{160+2\cdot 49}$
Space complexity	$\approx 2^{32}$	$\approx 2^{58}$	$\approx 2^{98}$

† That is the worst case, i.e. assuming all the subkeys are independent.

is that we do not need to keep separate counters for each key candidate. But, on the other hand, we need to store all the plaintext samples (therefore our space requirement: $N \approx \epsilon^{-2}$).

The 3.75 round linear relation (8) found for SAFER-K/-SK does not contradict the results in [3] (the 1.5 round relation (7)) because the former is non-homomorphic. Besides, the non-homomorphicity of linear relations (8)–(14) caused them to be key-dependent for the bit-mask approximations to hold, while (7) is key-independent. Another interesting observation is that relation (8) actually holds for any of the 128 possible S-box generators (of GF(257)), not only for 45 as used in SAFER-K/-SK and SAFER+. That is because, only few approximations are actually used, namely, the ones which explore the two LSB's in both the input and output masks. Therefore, changing the S-boxes' generator would not help protect these ciphers against our particular linear attack.

Nonetheless, our relation (14) is only valid for SAFER-K32 with generator $g = 11$. For the other seven possible generators of $GF(17)$ the linear relation (14) does not hold.

Table 2 compares the current analysis to other attacks on SAFER-K64. We conclude that the attack based on truncated differentials by Wu *et al.* [7] (which improves the original attack by Knudsen and Berson [17]) is still the best shortcut attack on SAFER-K64. Moreover, while differential attacks are typically chosen plaintext attacks, they can be converted to known plaintext attacks (see Biham and Shamir [6, p. 31]).

Theoretically it was predicted that for five rounds one key in 1024 is 'weak' (restrictions (9)), which means that the relation (8) with theoretical bias 2^{-37} can actually be used, restricted to weak keys, with bias 2^{-29}. Nonetheless, practical implementations of the attack show that only one out of eight keys is actually weak (only three of the subkey bits K_2^4, K_2^8, K_9^2, at the beginning and end of the linear hull need to have a certain bit-pattern). This may be another consequence of the linear-hull effect.

Table 2. Plaintext requirements of DC attacks by Knudsen–Berson (KB) and Wu, Boa, Deng and Ye (WBDY) and LC attacks on SAFER-K64

	Differential (chosen/known texts)			Linear (known texts)	
#rounds	KB [17]	WBDY [7]		Harpes [3]	this paper
	chosen	chosen	known		(weak keys)
2	—	—	—	$\approx 2^{15.2}$	$\approx 2^{8}$ †
3	—	—	—	$(> 2^{64})$	$\approx 2^{12}$ ‡
4	—	—	—	—	$\approx 2^{28}$ §
5	$\approx 2^{45}$	$\approx 2^{38}$	$\approx 2^{51}$	—	$\approx 2^{58}$
6	$(> 2^{64})$	$\approx 2^{53}$	$\approx 2^{59}$	—	$(> 2^{64})$
7	—	$(> 2^{64})$	$(> 2^{64})$	—	—

† Any homomorphic-only approximation can be used here
‡ This approximation comes from the first 1.75 rounds of relation (8)
§ This approximation comes from the first 2.75 rounds of relation (8)

The main conclusion however is that, while the analysis of Harpes *et al.* could be improved, linear cryptanalysis does not seem a serious threat, even to SAFER-K64 with its nominal number of rounds.

The main contribution of this paper towards better block cipher design is the issue of key-dependency in linear cryptanalysis, through non-homomorphic bitmasks. This improved on previous linear attacks on all SAFER family members by specifying linear relations valid for particular key classes; this analysis can have the same effect on other similar designs.

Acknowledgments. It is a pleasure to acknowledge the fruitful discussions with Johan Borst and Vincent Rijmen during this research and the preparation of all previous drafts of this paper.

References

1. Bluetooth Specification version 1.0B. Available at `http://www.bluetooth.com/link/spec/bluetooth_b.pdf`
2. Brincat, K., Meijer, A., "On the SAFER cryptosystem," *Cryptography and Coding, Proceedings of 6th IMA Conference, LNCS 1355,* M. Darnell, Ed., Springer-Verlag, 1997, pp. 59-68.
3. C. Harpes, *"Cryptanalysis of Iterated Block Ciphers,"* ETH series in Information Processing, J.L. Massey, Ed., Vol. 7, Hartung-Gorre Verlag, Konstanz, 1996.
4. C. Harpes, G. Kramer, J.L. Massey, "A generalization of linear cryptanalysis and the applicability of Matsui's piling-up lemma," *Advances in Cryptology, Proceedings Eurocrypt'95, LNCS 921*, L.C. Guillou and J.-J. Quisquater, Eds., Springer-Verlag, 1995, pp. 24–38.
5. D. Wagner, "The boomerang attack," *Fast Software Encryption, LNCS 1636*, L.R. Knudsen, Ed., Springer-Verlag, 1999, pp. 201–214.

6. E. Biham, A. Shamir, *"Differential Cryptanalysis of the Data Encryption Standard,"* Springer-Verlag, 1993.
7. H. Wu, F. Bao, R.H. Deng, Q.-Z. Ye, "Improved truncated differential attacks on SAFER," *Advances in Cryptology, Proceedings Asiacrypt'98, LNCS 1514*, K. Ohta, D. Pei, Eds., Springer-Verlag, 1998, pp. 133–147.
8. J. Borst, B. Preneel, J. Vandewalle, "Linear Cryptanalysis of RC5 and RC6," *Fast Software Encryption, LNCS 1636*, L.R. Knudsen, Ed., Springer-Verlag, 1999, pp. 16–30.
9. J. Kelsey, B. Schneier, D. Wagner, "Key schedule weaknesses in SAFER+," *Proceedings 2nd Advanced Encryption Standard Candidate Conference*, March 22–23, 1999, Rome (I), pp. 155–167.
10. J.L. Massey, G.H. Khachatrian, M.K. Kuregian, *"Nomination of SAFER+ as candidate algorithm for the Advanced Encryption Standard (AES),"* June 12, 1998. Available at `http://www.ii.uib.no/~larsr/aes.html`
11. J.L. Massey, "SAFER-K64: a byte-oriented block ciphering algorithm," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 1–17.
12. J.L. Massey, "SAFER-K64: one year later," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 212–241.
13. J.L. Massey, "Strengthened key schedule for the cipher SAFER," *posted to the USENET newsgroup sci.crypt*, September 1995. Available at `ftp://ftp.cert.dfn.de/pub/tools/crypt/SAFER/`
14. K. Nyberg, "Linear approximation of block ciphers," *Advances in Cryptology, Proceedings Eurocrypt'94, LNCS 950*, A. De Santis, Ed., Springer-Verlag, 1995, pp. 439–444.
15. L.R. Knudsen, "A key schedule weakness in SAFER-K64," *Advances in Cryptology, Proceedings Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 274–286.
16. L.R. Knudsen, "Why SAFER K changed its name," *Technical Report LIENS 96-13*, Laboratoire d'Informatique, Ecole Normale Supérieure, Paris, France, April 1996. Available at `http://www.ii.uib.no/~larsr/aes.html`
17. L.R. Knudsen, T.A. Berson, "Truncated differentials of SAFER," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 15–26.
18. M. Matsui, "Linear cryptanalysis method for DES cipher," *Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 386–397.
19. M. Matsui, A. Yamagishi, "A new method for known plaintext attack on FEAL cipher," *Advances in Cryptology, Proceedings Eurocrypt'92, LNCS 658*, R.A. Rueppel, Ed., Springer-Verlag, 1993, pp. 81–91.
20. S. Murphy, "An analysis of SAFER," *Journal of Cryptology*, Vol. 11, No. 4, 1998, pp. 235–251.
21. S. Vaudenay, "On the need for multipermutations: Cryptanalysis of MD4 and SAFER," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 286–297.

A A Ciphertext-Only Attack

In all previous linear attacks we did not make any assumption on the plaintext distribution. In many cases the plaintext consists (mostly) of printable ASCII characters, that is, characters with values between 20_x and $7E_x$. Matsui developed for this case a ciphertext only attack on DES with a reduced number of rounds [18]. For SAFER-K/SK, there exist linear hulls which allow for a 3-round ciphertext-only attack when the most significant bit of all plaintext bytes are equal to zero. One such hull with 2.25 rounds and $\mathcal{S} = 2$ is

$$\begin{aligned}&(\mathtt{0000000000000080_x}\ ,\mathtt{0200000000000000_x}\ ,2^{-6})\ \text{(one-round)}\\&(\mathtt{0200000000000000_x}\ ,\mathtt{0202020202020202_x}\ ,2^{-9})\ \text{(one round)}\\&(\mathtt{0202020202020202_x}\ ,\mathtt{0202020202020202_x}\ ,2^{-5})\ \text{(subkey quarter-round)}\end{aligned} \tag{20}$$

which has bias $\epsilon_4^* = 2^{-18}$. The key dependency conditions for the validity of (20) in a 0.75R attack on three rounds of SAFER-K/-SK are

$$\mathrm{LSB}(K_4^1, K_5^2, K_5^3, K_5^6, K_5^7) = 0\ , \tag{21}$$

The actual bias of (20) is therefore $\epsilon_4 = 2^{-13}$.

Let $P_1, \ldots, P_8$ denote the plaintext bytes, $D_1, \ldots, D_8$ the result of applying the inverse of the (unkeyed) PHT layer to the ciphertext bytes, and $X(.)$ and $L(.)$ the S-boxes. We use the following linear relation based on (20):

$$\begin{aligned}P_8\cdot\mathtt{80}\oplus L(D_1\boxminus K_6^1)\cdot\mathtt{02}\oplus X(D_2\oplus K_6^2)\cdot\mathtt{02}\oplus\\ X(D_3\oplus K_6^3)\cdot\mathtt{02}\oplus L(D_4\boxminus K_6^4)\cdot\mathtt{02}\oplus\\ L(D_5\boxminus K_6^5)\cdot\mathtt{02}\oplus X(D_6\oplus K_6^6)\cdot\mathtt{02}\oplus\\ X(D_7\oplus K_6^7)\cdot\mathtt{02}\oplus L(D_8\boxminus K_6^8)\cdot\mathtt{02}\oplus = K_i\cdot\mathit{\Gamma} K_i\ .\end{aligned} \tag{22}$$

If the plaintext is composed mostly of ASCII characters then equation (22) reduces to

$$\begin{aligned}L(D_1\boxminus K_6^1)\cdot\mathtt{02}\oplus X(D_2\oplus K_6^2)\cdot\mathtt{02}\oplus\\ X(D_3\oplus K_6^3)\cdot\mathtt{02}\oplus L(D_4\boxminus K_6^4)\cdot\mathtt{02}\oplus\\ L(D_5\boxminus K_6^5)\cdot\mathtt{02}\oplus X(D_6\oplus K_6^6)\cdot\mathtt{02}\oplus\\ X(D_7\oplus K_6^7)\cdot\mathtt{02}\oplus L(D_8\boxminus K_6^8)\cdot\mathtt{02}\oplus = K_i\cdot\mathit{\Gamma} K_i\ .\end{aligned} \tag{23}$$

keeping the same bias $\epsilon_4 = 2^{-13}$ because $P_8 \cdot \mathtt{80} = 0$ has bias $\epsilon_5 = 2^{-1}$.

The actual plaintext does not need to be composed of ASCII only characters, like in .HTML files. Some experiments show that even .JPG, .MP3, and .WAV files contain some small bias in the most significant bit of each byte, like $\epsilon_5 = 2^{-10}$; combined with the bias of (20) this results in a linear hull with bias $\epsilon_4 = 2^{-22}$ requiring about $N = 2^{44}$ ciphertext (only) blocks. Note that some popular (UNIX) file compression utilities like "compress" and "gzip" can destroy the redundancy of the MSB byte in ASCII files, but apparently they cannot destroy the bias of .JPG, .MP3 or .WAV files. Other possible biased distributions of (combinations of) plaintext bits can also be explored, that is, there is no need to consider only the most significant bit.

A Chosen-Plaintext Linear Attack on DES

Lars R. Knudsen and John Erik Mathiassen

Department of Informatics, University of Bergen, N-5020 Bergen, Norway
{lars.knudsen,johnm}@ii.uib.no

Abstract. In this paper we consider a chosen-plaintext variant of the linear attack on DES introduced by Matsui. By choosing plaintexts in a clever way one can reduce the number of plaintexts required in a successful linear attack. This reduces the amount of plaintexts to find key bits to a factor of more than four compared to Matsui's attack. To estimate the probabilities of success in the attack we did extensive experiments on DES reduced to 8 and 12 rounds. We believe that the results in this paper contain the fastest attack on the DES reported so far in the open literature. As an example, one attack needs about 2^{42} chosen texts, finds 12 bits of key information and succeeds with a probability of about 86%. An additional 12 key bits can be found by similar methods. For comparison, Matsui's attack on the DES needs about 2^{44} known texts, finds 13 bits of the key and succeeds with a probability of 78%. Of independent interest is a new approach searching for "pseudo-keys", which are secret key bits added an unknown but fixed value. These bits can be used to find the secret key bits at a later stage in the analysis.

1 Introduction

The DES is one of the most important cryptosystems that has been around in the open literature. Although it has seen the end of its days, this is mainly because of the short keys in the algorithm and not because any damaging intrinsic properties have been detected. In fact, today, about 25 years after the development of the DES, the most realistic attack is still an exhaustive search for the key. Several attacks have been developed which can find a DES-key faster than this, but all attacks reported require a huge amount of known or chosen plaintext-ciphertext pairs.

In 1992 Matsui introduced the linear cryptanalytic attack by applying it to FEAL [6] and one year later to the DES [3]. His attack on the DES using 2^{44} known texts, finds 13 bits of the key and succeeds with a probability of 78%. An additional 13 key bits can be found by a similar method. Subsequently, the remaining 30 bits can be found by exhaustive search. In [4] Matsui also considers "key-ranking", where one considers the attack successful if the correct key is amongst the q most likely keys. Clearly, with key-ranking the success rates will be higher or the text requirements decrease for the same success probability. If we assume that the number of key bits found by the attack is k, one does an exhaustive search for the remaining $56 - k$ bits for each of the q candidates of the first k bits. Thus, key-ranking can be used to decrease the number of texts needed but wil increase the computational effort in the final key search. Matsui implemented this attack in January 1994 and successfully recovered one DES-key after the encryption of 2^{43} plaintext blocks.

In this paper, if not stated otherwise, all the reported success rates are measured as the number of times the correct value of the key is the most likely candidate suggested by the attack. Clearly, with key-ranking the success rates will be higher.

In the year following Matsui's publications, several reports were publicised which modify and improve on his results e.g., [1,2,7,8]. However until now these approaches have led to only small improvements for attacks on the DES. One exception is the chosen-plaintext differential-linear attack which led to a big reduction in the number of texts needed, however the attack as reported is applicable to only up to 8 rounds of the DES.

In this paper another chosen-plaintext variant of the linear attack on the DES is studied. It is shown that in this scenario it is possible to reduce the number of required texts (the main obstacle in the attack) to a factor of more than four less than that required by Matsui's attack. We use what we believe is a new idea in cryptanalytic attacks, namely in a first-phase of the attack we search for "pseudo-keys", which are the secret keys added some unknown, but fixed value. In a later stage these pseudo-key bits can be used to reduce an exhaustive key search.

In § 2 we introduce the most important concepts and results of the linear attack on the DES. In § 3 we outline three possible chosen-plaintext variants. All but the second variant can be used to attack the DES up to 16 rounds. The second one is limited to attack DES up to 12 rounds.

2 Linear Cryptanalysis on DES

In linear cryptanalysis one tries to find probabilistic linear relations between the plaintext P, the ciphertext C, and the key K. The easiest way to obtain this is to look for one-round linear relations and use these iteratively to obtain relations over more rounds. First we consider one-round relations. In the following let C_i denote the ciphertext after i rounds of encryption. Then a linear expression in the ith round has the following form.

$$(C_i \cdot \alpha) \oplus (C_{i+1} \cdot \beta) = (K_i \cdot \gamma), \tag{1}$$

where α, β and γ are bit-masks and '$\cdot$' is a bit-wise dot product operator. The masks are used to select the bits of a word used in the linear relation. The bit masks (α, β) are often called one-round linear characteristics. Since the key K_i is a constant, one looks at the probability p_i that the left side of (1) equals 0 or 1. We denote by the bias, the quantity $|p_i - \frac{1}{2}|$. For the DES one can easily calculate all linear relations for the S-boxes, combine these and get all possible linear relations for one round of the cipher. Subsequently, one can combine the one-round relations to get linear relations for several rounds under the assumption of independent rounds. To calculate the probabilities one usually uses the Piling-up Lemma:

Lemma 1. *Let Z_i, $1 \leq i \leq n$ be independent random variables in $\{0,1\}$. If $Z_i = 0$ with probability p_i we have*

$$Pr(Z_1 \oplus Z_2 \oplus \ldots \oplus Z_n = 0) = \frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} (p_i - \frac{1}{2}) \tag{2}$$

For most ciphers the one-round linear relations involved in a multi-round relation are not independent. For the DES the relations are dependent, but our experiments, as well as Matsui's experiments [3,4], show that Piling-up Lemma gives a good approximation for the DES.

For the DES, Matsui has provided evidence [5] that the best linear characteristics over 14 rounds or more are obtained by iterating 4-round characteristics.

Four-round iterative characteristics. The four-round characteristic used in Matsui's attack on the DES is shown in Fig. 1. Let X_i denote the input to the F-function in the ith round. For convenience we shall write $F(X_i)$ instead of $F(X_i, K_i)$. The masks A, D and B are chosen to maximise the probabilities of following linear relations.

$$\begin{aligned} F(X_1) \cdot A &= X_1 \cdot D \qquad \text{with prob. } p_1, \\ F(X_3) \cdot B &= X_3 \cdot D \qquad \text{with prob. } p_3, \ \text{ and} \\ F(X_2) \cdot D &= X_2 \cdot (A \oplus B) \qquad \text{with prob. } p_2. \end{aligned}$$

Then it follows from the Piling-Up Lemma and by easy calculations that the relation $(X_0 \cdot A) \oplus (X_4 \cdot B) = 0$ holds with probability

$$P_L = \frac{1}{2} - 4(p_1 - \frac{1}{2})(p_2 - \frac{1}{2})(p_3 - \frac{1}{2}).$$

This 4-round characteristic can be iterated to a 14-round characteristic, which in a short, space-consuming notation is

```
 2: -  -  -
 3: A ← D
 4: D ← A⊕B
 5: B ← D
 6: -  -  -
 7: B ← D
 8: D ← A⊕B
 9: A ← D
10: -  -  -
11: A ← D
12: D ← A⊕B
13: B ← D
14: -  -  -
15: B ← D
```

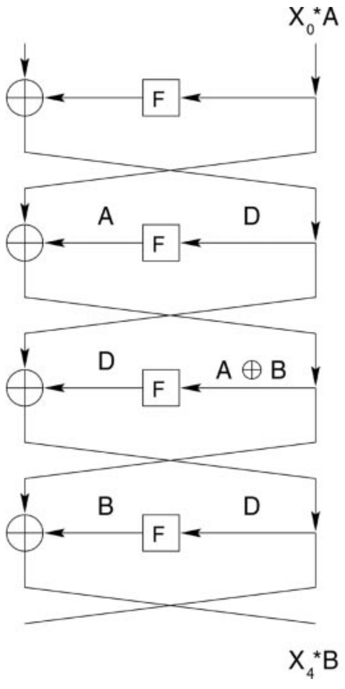


Fig. 1. 4-round linear characteristic of DES.

Here 'n:' denotes that the expression occurs in round no. n and '- - -' means that no approximation is made in the round. Notice that A and B are interchanged for every 4-round iteration. This leads to the equation for 16-round DES:

$$(P^L \cdot A) \oplus (F(P^R, K_1^*) \cdot A) \oplus (C^L \cdot D) \oplus (F(C^R, K_{16}^*) \cdot D) \oplus (C^R \cdot B) = 0 \quad (3)$$

The probability for this equation is

$$P_L^{15} = \frac{1}{2} + 2^{14-1} \prod_{i=2}^{15} (p_i - \frac{1}{2}) \quad (4)$$

where

$$\begin{aligned} p_i &= 1, \quad i \in \{2, 6, 10, 14\}, \\ p_i &= 42/64, \quad i \in \{3, 9, 11\}, \\ p_i &= 30/64, \quad i \in \{4, 8, 12\}, \\ p_i &= 12/64, \quad i \in \{5, 7, 13, 15\} \end{aligned}$$

For the correct guesses of K_1^* and K_{16}^* the equation (3) will have probability P_L^{15}. For other keys, the equation will look random. In the attack one keeps a counter

Table 1. Complexities of Matsui's linear attack on 8-round DES and full DES, where 13 key bits are found.

	8-round DES			16-round DES		
Plaintexts	2^{18}	2^{19}	2^{20}	2^{43}	2^{44}	2^{45}
Success rate	49.4%	93.2%	100%	32.5%	77.7%	99.4%

for each value of the secret key $(K_1^{'}, K_{16}^{'})$ which keeps track of the number of times the left side of the equation is 0. With N (P, C)-pairs, the key $(K_1^{'}, K_{16}^{'})$ with counter value T farthest from $\frac{N}{2}$ is taken as the correct value of the key. The sum of the key bits involved in the approximation can also be found [3]. The probability of success can be calculated by a normal approximation of the binomial distribution. Under the assumptions that $|P_L - \frac{1}{2}|$ is small, it can be shown that if one chooses $N = (P_L - \frac{1}{2})^{-2}$, one gets a probability of 97.72% that the value T of the counter for the correct value of key is more than $N/2$ when $P_L > \frac{1}{2}$ and less otherwise. However, there will be noise from the wrong keys also which have to be considered. It has been conjectured and confirmed by computer experiments that the left side of (3) will look random when wrong values of the keys are used [3]. It was also estimated by experiments that the complexity Np for the attack on DES is

$$Np \approx c|p_L - \frac{1}{2}|^{-2}$$

where $c \leq 8$. To confirm the theory we implemented tests on DES reduced to 8 rounds. The equation (3) for 8 rounds is the same as for 16 rounds except for the index of the key in the last round. For 8 rounds $Np = c \times 0.95 \times 2^{16}$. Our experimental results for 8-rounds DES can be found in Table 1.

This attack finds 13 bits of the key. It is possible to find a total of 26 key bits by using the same linear characteristic on the decryption operation. In this case the probabilities in Table 1 must be squared.

The complexity of the attack on the DES can be estimated from the complexity of the attack on 8-round DES. If one lets the complexity for the attack on 8-round DES be Np_8, the expected complexity Np_{16} for 16-round DES can be calculated such that the success probabilities are approximately the same. The formula is [4]

$$Np_8 = Np_{16}|P_{L16} - 1/2|^2/|P_{L8} - 1/2|^2.$$

With $Np_{16} = 2^{45}$ one gets

$$Np_8 = 2^{45} \times |1.19 \times 2^{-21}|^2/|1.95 \times 2^{-9}|^2 = 1.49 \times 2^{19}.$$

Thus, the success probability of the attack on 8-round DES with $N = 1.5 \times 2^{19}$ will be the same as the attack on 16-round DES with $N = 2^{45}$. The estimates of the complexity of the linear attack by Matsui, where 13 key bits are found, can be found in Table 1.

Table 2. Complexities of the first chosen-plaintext variant of the linear attack on 8-round and 12-round DES finding 7 key bits.

	8-round DES			12-round DES		
Plaintexts	2^{18}	2^{19}	2^{20}	2^{28}	2^{29}	2^{30}
Success rate	68%	99%	100%	46%	72%	94%

3 Chosen-Plaintext Attacks

In this section we consider chosen-plaintext variants of the linear attack on the DES. The time complexity of the reported attacks is always less than the data complexity, that is, the number of needed texts, and is therefore ignored in the following.

3.1 First Attack

A first chosen-plaintext extension is an attack where one does not search for the key in the first round, merely for six bits of the key in the last round, but the bias for the equation remains the same. Since the noise of 63 wrong keys is less than of 4095 wrong keys, the attack is expected to be of lower complexity than that of Matsui. The trick is that we fix the six input bits to the active S-box in the first round. Then any output mask of that function is a constant 0 or 1 with bias $\frac{1}{2}$. One then considers the following equation:

$$(P^L \cdot A) \oplus (C^L \cdot D) \oplus (F(C^R, K_n^*) \cdot D) \oplus (C^R \cdot B) = 0 \qquad (5)$$

For all guesses of the key K_n one counts the number of times the left side of the equation equals zero. Hopefully for the correct value one gets a counter with a value that differs from the mean value $\frac{N}{2}$ more than for all other counters. With a sufficient number of texts (N) this will work. Also, one can determine a seventh key bit from the bias of the equation, when searching for the rest of the key bits. The estimated number of plaintexts required, Np, is less, although only slightly less, than for Matsui's attack. The complexities of the attack on 8-round and 12-round DES are given in Table 2.

3.2 Second Attack

In addition to fixing the six bits of the input to the active S-box (no. 5) in the first round, one can try to do the same for a possible active S-box in the second round. For the 14-round characteristic used by Matsui there is no active S-box in the second round. However, if one takes the first 13 rounds of this characteristic and uses these in the rounds 3 to 15 one gets a single active S-box in both the first and second rounds. The we get the following picture.

```
 3: -  -  -
 4: A ← D
 5: D ← A⊕B
 6: B ← D
 7: -  -  -
 8: B ← D
 9: D ← A⊕B
10: A ← D
11: -  -  -
12: A ← D
13: D ← A⊕B
14: B ← D
15: -  -  -
```

Now we can fix the inputs to all S-boxes in the first round which output bits are input to the active S-box in the second round. To achieve this we need to fix the inputs to six S-boxes in the first round, totally 28 bits, and to fix six bits of the left half of the plaintext. Thus one needs to fix 34 bits of all plaintexts which is illustrated in Figure 2. This also means that an attacker only has 30 bits to his disposal in an attack. However, it also means that there is one round less to approximate and one would expect higher success rates.

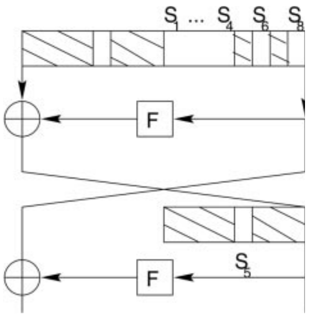


Fig. 2. The first two rounds in the linear characteristic. The bits in the striped blocks vary under the control of the attack. The bits in the white blocks are fixed.

The equation to solve in the key search is the following.

$$(P^R \cdot A) \oplus (C^L \cdot B) \oplus (F(C^R, K_n^*) \cdot B) = 0 \tag{6}$$

In this case we are able to find only six bits in the last-round key K_n, plus one key-bit from the sign of the counter T minus $\frac{N}{2}$. The probability calculation for the attack on 16-round DES is

$$P_L = \frac{1}{2} + 2^{13-1} \prod_{i=3}^{15} (p_i - \frac{1}{2}). \tag{7}$$

Table 3. Complexities of the second chosen-plaintext variant of the linear attack on 8-round DES, where we found 7 key bits.

Plaintexts	2^{16}	2^{17}	2^{18}
Success rate	78%	98%	100%
Success rate 2	90%	100%	100%

The number of chosen plaintexts needed is $Np = c|P_L - \frac{1}{2}|^{-2}$. This is a factor of $(\frac{8}{5})^2 \approx 2.6$ less than in the previous attack. By interchanging the rounds in the characteristic one can also solve for the equation

$$(P^R \cdot B) \oplus (C^L \cdot A) \oplus (F(C^R, K_{16}^*) \cdot A) = 0, \tag{8}$$

where we just flip the characteristic. Note that the involved active S-boxes and key bits are the same as for the first characteristic. This increases the success rate because for the correct key K_n we have the same sign of the bias in the two expressions. Our test results on 8-round DES of the success rate where we use one equation is shown in the first line of Table 3 and the second line is the case where we use both equations (6) and (8).

3.3 Third Attack

One problem with the previous variant is that there is a limit of 2^{30} possible texts to be used in an attack, and the attack will not be applicable to 16-round DES. In the following it is shown how more texts can be made available. This variant attack is based on two methods that we will introduce.

Pseudo-keys: In the first method we fix the same 28 bits in the right halves of the plaintexts as before. This gives a constant output for the six desirable bits which is output from the first round function and which affect the input to the active S-box in the second round. Let us denote these six bits by y_1. But where before we fixed also the six bits of the left halves of the plaintext that affect the active S-box in the second round we will now allow these to change. If we denote by K_2 the key affecting the active S-box in the second round, we define a "pseudo key" $K_2^{'} = K_2 \oplus y_1$. This allows us to search for and find six bits of $K_2^{'}$ in addition to the six bits of K_n. At this point we are able to generate 2^{36} different plaintexts with the desired property. We then try to solve the following equation:

$$(P^R \cdot A) \oplus (F(P^L, K_2^{'*}) \cdot B) \oplus (C^L \cdot B) \oplus (F(C^R, K_n^*) \cdot B) = 0 \tag{9}$$

When the attack terminates, the correct key K_2 can be determined from $K_2^{'}$ by simply adding $y_1 = F(P^R, K_1)$ when searching exhaustively for the remaining key bits of the key K and thereby K_1. There is also some overlap between the key bits of K_1 and the six bits of K_2. This must be taken into consideration when searching for the key.

Additional plaintexts: Here we show how to be able to control an additional six bits of the plaintexts. These must be bits in the right halves of the plaintexts, since all bits in the left halves are assumed to be under control of the attack already. However, this creates two problems. First, if we are going to vary some of the input bits to an S-box in round one, we also change the output bits of the round function, which were assumed to be fixed above. Second, changing the input bits to one S-box might affect the neighboring S-boxes, as these have overlapping, common input bits. However, the S-boxes 5 and 7 are not assumed to have a fixed input in the above attacks. (This allowed us to control and vary the middle two input bits to both S-boxes in the above attack.) Thus, if we vary the six bits input to S-box 6 in the first round, the second problem is overcome. Totally this gives the attack control over 2^{42} plaintexts. The first problem can be overcome by searching also for the affected six key bits entering S-box 6 in the first round. Note that when we vary the inputs to this S-box one of the bits of y_1 will vary. For each guess of the key to S-box 6 in the first round, we take this one bit into account when searching for K_2'. Fig. 3 illustrates which bits are fixed in the attack and which bits are not. The equation for this attack is

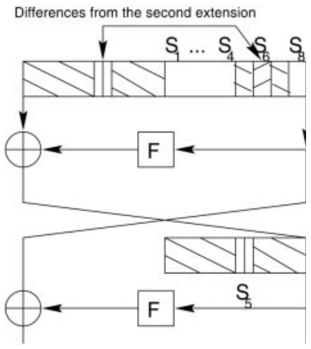


Fig. 3. The first two rounds in the third attack. The inputs to the S-boxes 1,2,3,4, and 8 in the first round are fixed.

$$(P^R \cdot A) \oplus (C^L \cdot B) = (F(P^L, F(P^R, K_1^*) \oplus K_2^*) \cdot A) \oplus (F(C^R, K_n^*) \cdot B) \quad (10)$$

which has the same bias as (6). For the correct values of the three keys K_1^*, K_2^* and K_n^* observe that the equation will have probability P_L (of (7)). For wrong values the equation will look random. There are three bits in K_n and one in K_2 which overlap with key bits in K_1. Potentially the attack could find 15 key bits.

However, after implementing this attack we found that it is difficult to determine the correct value of two bits of the key K_1. The reason for this is than on the average in 50% of the cases, if the one bit of $F(P^R, K_1^*)$ that is input to S-box 5 in the second round is wrong, the masked output from S-box 5 in round 2 will still be correct. This has the effect that determining the two bits of K_1

Table 4. Complexities of the third chosen-plaintext variant of the linear attack on 8-round and 12-round DES. Here we found 12 key bits.

	8-round DES		12-round DES	
Plaintexts	2^{16}	2^{17}	2^{28}	2^{29}
Success rate	51%	94%	28%	76%

Table 5. Complexities of the known-plaintext and chosen-plaintext linear attacks on the DES. Matsui finds 13 key bits and we find 12 key bits.

	Matsui's attack			Our attack		
Plaintexts	2^{43}	2^{44}	2^{45}	2^{40}	2^{41}	2^{42}
Success rate	32%	78%	99%	6%	32%	86%

which do not overlap with bits in K_2 and K_{16} requires much effort. Because of this it is equally difficult to determine the third least significant bit in K_2. The attack finds eleven key bits much faster than all fourteen key bits. The attack can also find a 12th key bit in the similar way as in the previous attack. Simply look at the sign of the bias $|P_L - \frac{1}{2}|$ and compare with the bias of the key guessed. We implemented 100 tests with randomly chosen keys for 8-round DES and 50 similar tests for 12-round DES. The results can be found in Table 4. Thus, this variant of the attack has a poorer performance than in the previous attack, the advantage is that more plaintexts are available and a potential attack on 16-round DES is emerging.

We may estimate the success rate for 16-round DES as follows. One can calculate the expected number of plaintexts for 8-round DES, Np_8, and for 16-round DES, Np_{16}, which will have the same success rate. The ratio is the same as for Matsui's attack, because in these two attacks the bias differ with the same factor for both 8 and 16 rounds. E.g., we have that the success rate for 16-round DES using 2^{42} texts is the same as for the attack on 8-round DES with

$$Np_8 = 2^{42} \times |1.91 \times 2^{-21}|^2 / |1.56 \times 2^{-8}|^2 = 1.49 \times 2^{16}$$

texts. Similar, one gets from the attack on 12-round DES that with

$$Np_{12} = 2^{42} \times |1.91 \times 2^{-21}|^2 / |1.21 \times 2^{-14}|^2 = 1.25 \times 2^{29}$$

texts the success rate is the same as for the attack on 16-round DES with 2^{42} texts. From the experiments on 8-round and 12-round DES one gets the complexities of the chosen-plaintext linear attack on the DES of Table 5.

In total the attack finds 12 bits of key information. By repeating the attack on the decryption operation of DES an additional 12 bits of key information can be found. Subsequently, it is easy to find the remaining 32 bits by an exhaustive search. Using key-ranking the reported rates of success will be even higher. As an example, in the tests of Table 4 on 8-round DES using 2^{16} texts, the correct key appeared as one of the 8 highest ranked keys in 90 of the 100 tests, and using 2^{17} texts, the correct key was ranked 2,2,2,3,3, and 4 in the tests where it was not the first.

4 Conclusion

In this paper we presented what we believe is the fastest attack reported on the DES. The attack requires 2^{42} chosen plaintexts and finds 12 bits of the secret key with a probability of success of 86%. This should be compared to Matsui's attack, which finds one more key bit using a factor of four more plaintexts. Subsequently, in our attack, the remaining 44 bits of a DES key can be found by an exhaustive search or alternatively, an additional 12 key bits can be found by repeating the attack on the decryption routine. A new approach in our attacks is the search for "pseudo-key bits", which are secret key bits added with some unknown but fixed value. In a subsequent key search these pseudo-keys can be used to find real key bits. This approach might be applicable to similar attacks on other ciphers.

References

1. B.S. Kaliski and M.J.B. Robshaw. Linear cryptanalysis using multiple approximations. In Y. Desmedt, editor, *Advances in Cryptology: CRYPTO'94, LNCS 839*, pages 26–39. Springer Verlag, 1994.
2. L.R. Knudsen and M.P.J. Robshaw. Non-linear approximations in linear cryptanalysis. In U. Maurer, editor, *Advances in Cryptology: EUROCRYPT'96, LNCS 1070*, pages 224–236. Springer Verlag, 1996.
3. M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor, *Advances in Cryptology - EUROCRYPT'93, LNCS 765*, pages 386–397. Springer Verlag, 1993.
4. M. Matsui. The first experimental cryptanalysis of the Data Encryption Standard. In Y.G. Desmedt, editor, *Advances in Cryptology - CRYPTO'94, LNCS 839*, pages 1–11. Springer Verlag, 1994.
5. M. Matsui. On correlation between the order of S-boxes and the strength of DES. In A. De Santis, editor, *Advances in Cryptology - EUROCRYPT'94, LNCS 950.* Springer Verlag, 1995.
6. M. Matsui and A. Yamagishi. A new method for known plaintext attack of FEAL cipher. In R. Rueppel, editor, *Advances in Cryptology - EUROCRYPT'92, LNCS 658*, pages 81–91. Springer Verlag, 1992.
7. T. Shimoyama and T. Kaneko. Quadratic relation of s-box and its application to the linear attack of full round DES. In H. Krawczyk, editor, *Advances in Cryptology: CRYPTO'98, LNCS 1462*, pages 200–211. Springer Verlag, 1998.
8. S. Vaudenay. An experiment on DES - statistical cryptanalysis. In *Proceedings of the 3rd ACM Conferences on Computer Security, New Delhi, India*, pages 139–147. ACM Press, 1995.

Provable Security against Differential and Linear Cryptanalysis for the SPN Structure

Seokhie Hong[1]*, Sangjin Lee, Jongin Lim, Jaechul Sung, Donghyeon Cheon, and Inho Cho

Center for Information and Security Technologies(CIST), Korea University, Seoul, KOREA, hsh@semi.korea.ac.kr

Abstract. In the SPN (Substitution-Permutation Network) structure, it is very important to design a diffusion layer to construct a secure block cipher against differential cryptanalysis and linear cryptanalysis. The purpose of this work is to prove that the SPN structure with a maximal diffusion layer provides a provable security against differential cryptanalysis and linear cryptanalysis in the sense that the probability of each differential (respectively linear hull) is bounded by p^n (respectively q^n), where p (respectively q) is the maximum differential (respectively liner hull) probability of n S-boxes used in the substitution layer. We will also give a provable security for the SPN structure with a semi-maximal diffusion layer against differential cryptanalysis and linear cryptanalysis.

1 Introduction and Motivation

The Feistel structure has been used widely in the iterated block cipher. In this structure, the input to each round is divided into two halves. The right half is transformed by some nonlinear function and then xored to the left half and the two halves are swapped except for the last round. On the other hand, the SPN structure is designed using round function on the whole data block. Nowadays, the SPN structure is also attracting interest because it is highly parallelizable and easy to analyze the security against differential cryptanalysis(DC) and linear cryptanalysis(LC).

The most well known attacks on block ciphers are DC[1,2,3] and LC[6,7]. In DC, one uses characteristic which describes the behavior of input and output differences for some number of consecutive rounds. But it may not be necessary to fix the values of input and output differences for the intermediate rounds in a characteristic, so naturally the notion of differential was introduced[15]. The same statements can be applied to LC, so that of linear hull was introduced[11]. However it seems computationally infeasible to compute the maximum probabilities of differential and linear hull if the number of rounds increases.

* The authors wish to acknowledge the financial support of the Korea Research Foundation made in the program year of 1998.

In [9], K. Nyberg and L.R. Knudsen showed that the r-round differential probability is bounded by $2p^2$ if the maximal differential probability of round function is p and $r \geq 4$. Furthermore, the probability can be reduced to p^2 if the round function is bijective. These results provide a provable security for the Feistel structure against DC. M. Matsui proposed a new block cipher of a Feistel network, MISTY[8] for which security can be shown by the existing results for Feistel structures. The round function of MISTY is itself a Feistel network which is proven secure. From this round function with small S-boxes, he provided sufficiently large and strong S-boxes with proven security.

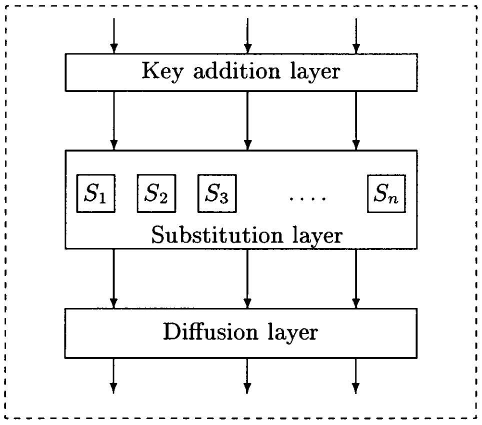


Fig. 1. One round of a SPN structure

In the SPN structure the diffusion layer provides an avalanche effect, both in the contexts of differences and linear approximation, so the notion of branch number was introduced[16]. The branch number of a diffusion layer has been determined to be very important. A cipher with the low branch number may have a fatal weakness even though a substitution layer consists of S-boxes resistant against DC and LC. In this paper we will give a provable security for the SPN structure with a maximal branch number by theorem 1.

This paper proceeds as follows; In section 2 we will introduce some notations and definitions. In section 3 a provable security for the SPN structure with a maximal diffusion layer against DC will be given. Provable security against LC will be given in section 4. Other results will be described in section 5.

2 Preliminaries

In this section we define some notations and definitions. Throughout this paper we consider an SPN structure with mn-bit round function. Let S_i be an $m \times m$

bijective S-box, i.e.,

$$S_i \;:\; \mathbb{Z}_2^m \to \mathbb{Z}_2^m \;\; (1 \le i \le n).$$

Definition 1. *For any given $\Delta x, \Delta y, \Gamma x, \Gamma y \in \mathbb{Z}_2^m$, the differential and linear probability of each S_i are defined as follows;*

$$DP^{S_i}(\Delta x \to \Delta y) = \frac{\#\{x \in \mathbb{Z}_2^m | S_i(x) \oplus S_i(x \oplus \Delta x) = \Delta y\}}{2^m}$$

$$LP^{S_i}(\Gamma x \to \Gamma y) = \left(\frac{\#\{x \in \mathbb{Z}_2^m | \Gamma x \cdot x = \Gamma y \cdot S_i(x)\}}{2^{m-1}} - 1 \right)^2$$

where $\Gamma x \cdot x$ denotes the parity of bitwise xor of Γx and x.

Definition 2. *The maximal differential and linear probability of S_i are defined by*

$$DP^{S_i}_{max} = \max_{\Delta x \neq 0, \Delta y} DP^{S_i}(\Delta x \to \Delta y)$$

and

$$LP^{S_i}_{max} = \max_{\Gamma x, \Gamma y \neq 0} LP^{S_i}(\Gamma x \to \Gamma y),$$

respectively.

In general, S_i is called strong if $DP^{S_i}_{max}$ and $LP^{S_i}_{max}$ are small enough and a substitution layer is called strong if $DP^{S_i}_{max}$ and $LP^{S_i}_{max}$ are small enough for all $1 \le i \le n$. Let us denote by p and q the maximal value of $DP^{S_i}_{max}$ and $LP^{S_i}_{max}$ for $1 \le i \le n$, respectively. That is,

$$p = \max_{1 \le i \le n} DP^{S_i}_{max}, \;\; q = \max_{1 \le i \le n} LP^{S_i}_{max}.$$

Even though p and q are small enough, this does not guarantee a secure SPN structure against DC and LC. Hence the role of the diffusion layer is very important. The purpose of the diffusion layer is to provide an avalanche effect, both in the contexts of differences and linear approximations.

Definition 3. *Differentially active S-box is defined as an S-box given a non-zero input difference and linearly active S-box as an S-box given a non-zero output mask value[5].*

The number of differentially active S-boxes has an effect on probabilities of differential characteristics or differentials. Hence the concept of active S-box plays an important role in giving a provable security for the SPN structure. Conversely differentially(resp. linearly) inactive S-boxes have a zero input xor(resp. output mask value). Consequently they have always a zero output xor(resp. input mask value) with probability 1.

Let $x = (x_1, \cdots, x_n)^t \in GF(2^m)^n$ then the Hamming weight of x is denoted by

$$Hw(x) = \#\{i | x_i \neq 0\}.$$

Note "Hamming weight of X" does not count the number of nonzero bits but count the number of non-zero m-bit characters.

Throughout this paper we assume that the round keys, which are xored with the input data at each round, are independent and uniformly random. By assumption on round keys, key addition layer in Fig.1 has no influence on the number of active S-boxes. Now we define a SDS function with three layer of substitution-diffusion-substitution as depicted in Fig.2.

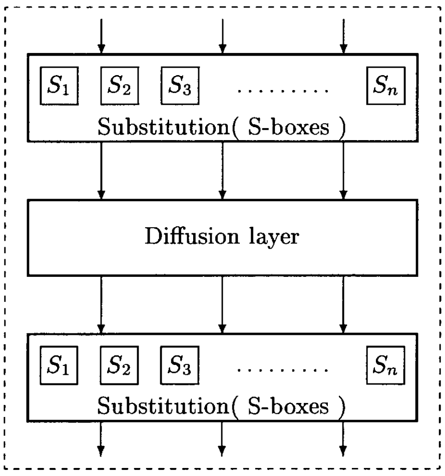


Fig. 2. SDS function

Denote diffusion layer of this SDS function by D, input difference by $\Delta x = x \oplus x^*$, output difference by $\Delta y = y \oplus y^* = D(x) \oplus D(x^*)$, and finally input mask value and output mask values by Γx and Γy, respectively. The minimum number of differentially and linearly active S-boxes of the SDS function are defined as follows;

$$n_d(D) = \min_{\Delta x \neq 0} Hw(\Delta x) + Hw(\Delta y)$$

and

$$n_l(D) = \min_{\Gamma y \neq 0} Hw(\Gamma x) + Hw(\Gamma y),$$

respectively[12]. $n_d(D)$ and $n_l(D)$ are lower bounds for the number of active S-boxes in two consecutive rounds of a differential characteristic and linear approximation, respectively. A diffusion layer is called **maximal** if the $n_d(D)$(equivalently $n_l(D)$) is $n+1$.

3 Provable Security against DC

In this section we will give a provable security for the SPN structure with a maximal diffusion layer against DC. Throughout this paper we assume that the diffusion layer D of the SDS function can be represented by an $n \times n$ matrix $M = (m_{ij})_{n\times n}$, where $m_{ij} \in GF(2^m)$. That is,

$$M = \begin{pmatrix} m_{11} & \cdots & m_{1n} \\ \vdots & \ddots & \vdots \\ m_{n1} & \cdots & m_{nn} \end{pmatrix}.$$

J. Daemen et. al [4] showed that, for the diffusion layer D, the relation between input difference(resp. output mask value) and output difference(resp. input mask value) is represented by the matrix M(resp.M^t). That is to say,

$$\Delta y = M\Delta x \ (resp.\ \Gamma x = M^t \Gamma y).$$

So we can redefine $n_d(D)$ and $n_l(D)$ as follows;

$$n_d(D) = \min_{\Delta x \neq 0} \{Hw(\Delta x) + Hw(M\Delta x)\},$$

$$n_l(D) = \min_{\Gamma y \neq 0} \{Hw(\Gamma y) + Hw(M^t \Gamma y)\}.$$

Hence we only need to investigate the matrix M to analyze the role of the diffusion layer D. Let us call M' an $s \times k$ submatrix of M if M' is of the following form;

$$M' = \begin{pmatrix} m_{i_1 j_1} & m_{i_1 j_2} & \cdots & m_{i_1 j_k} \\ m_{i_2 j_1} & m_{i_2 j_2} & \cdots & m_{i_2 j_k} \\ \vdots & \vdots & \ddots & \vdots \\ m_{i_s j_1} & m_{i_s j_2} & \cdots & m_{i_s j_k} \end{pmatrix}$$

Then we say that M contains M' as an $s \times k$ submatrix.

The following lemma shows the necessary and sufficient condition for a diffusion layer to be maximal.

Lemma 1. *Let M be the $n \times n$ matrix representing a diffusion layer D. Then $n_d(D) = n + 1$ if and only if the rank of each $k \times k$ submatrix of M is k for all $1 \leq k \leq n$.*

Proof Assume that $n_d = n + 1$ and there exists a $k \times k$ submatrix M_k of M such that the rank of M_k is less than k for some $1 \leq k \leq n$. Without loss of generality we may assume that

$$M_k = \begin{pmatrix} m_{11} & \cdots & m_{1k} \\ \vdots & \ddots & \vdots \\ m_{k1} & \cdots & m_{kk} \end{pmatrix}.$$

By assumption there exists $(x_1, \cdots, x_k) \neq (0, \cdots, 0)$ such that

$$\begin{pmatrix} m_{11} & \cdots & m_{1k} \\ \vdots & \ddots & \vdots \\ m_{k1} & \cdots & m_{kk} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{1}$$

Let $x = (x_1, \cdots, x_k, 0, \cdots, 0)^t$. By equation (1),

$$Mx = \begin{pmatrix} m_{11} & \cdots & m_{1k} & m_{ik+1} & \cdots & m_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{k1} & \cdots & m_{kk} & m_{kk+1} & \cdots & m_{kn} \\ m_{k+11} & \cdots & m_{k+1k} & m_{k+1k+1} & \cdots & m_{k+1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{n1} & \cdots & m_{nk} & m_{nk+1} & \cdots & m_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_k \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \delta_{k+1} \\ \vdots \\ \delta_n \end{pmatrix}. \tag{2}$$

By the definition of $n_d(D)$,

$$n_d(D) \leq Hw(x) + Hw(Mx) \leq k + n - k = n.$$

This is a contradiction to $n_d = n+1$. Therefore we obtained a sufficient condition.

Assume that the rank of each $k \times k$ submatrix of M is k for all $1 \leq k \leq n$ and $n_d < n+1$. Since $n_d < n+1$, there exists $x = (x_1, \cdots, x_n)^t \in GF(2^m)^n$ such that

$$Hw(x) + Hw(Mx) \leq n.$$

Without loss of generality we may assume that $x_1, \cdots, x_s$ are all nonzero and $x_j = 0$ for all $j > s$. Let $y = Mx$, then $Hw(y) \leq n - s$. In other words, the number of zero components in y is greater than or equal to s, so we can assume $y_{i_1} = \cdots = y_{i_s} = 0$. We can easily check equation (3).

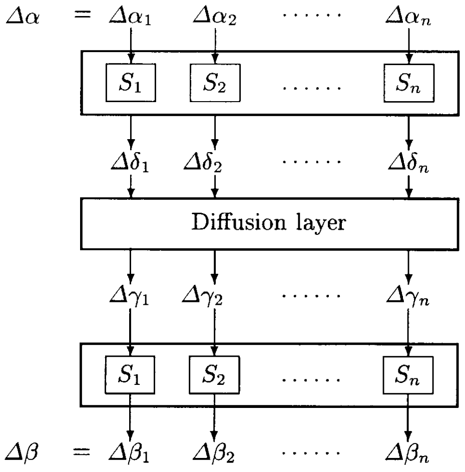
$$\begin{pmatrix} m_{i_1 1} & \cdots & m_{i_1 s} \\ \vdots & \ddots & \vdots \\ m_{i_s 1} & \cdots & m_{i_s s} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_s \end{pmatrix} = \begin{pmatrix} y_{i_1} \\ \vdots \\ y_{i_s} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3}$$

Hence we can get an $s \times s$ submatrix of M with rank less than s. It is a contradiction to the fact that the rank of each $k \times k$ submatix of M is k for all $1 \leq k \leq n$. ■

In [12], it was shown how a maximal diffusion layer over $GF(2^m)^n$ can be constructed from a maximum distance separable code. If $G_e = [I_{n\times n} B_{n\times n}]$ is the echelon form of the generator matrix of $(2n, n, n+1)$ RS-code, then

$$\begin{aligned} D : GF(2^m)^n &\to GF(2^m)^n \\ x &\mapsto Bx \end{aligned}$$

is a maximal diffusion layer by lemma 1.



$\Delta\alpha = \Delta\alpha_1 \quad \Delta\alpha_2 \cdots\cdots \Delta\alpha_n$
S_1 S_2 ...... S_n
$\Delta\delta_1 \quad \Delta\delta_2 \cdots\cdots \Delta\delta_n$
Diffusion layer
$\Delta\gamma_1 \quad \Delta\gamma_2 \cdots\cdots \Delta\gamma_n$
S_1 S_2 ...... S_n
$\Delta\beta = \Delta\beta_1 \quad \Delta\beta_2 \cdots\cdots \Delta\beta_n$

Fig. 3. Differential of SDS function

Consider the differential with input difference $\Delta\alpha = (\Delta\alpha_1, \cdots, \Delta\alpha_n)$ and output difference $\Delta\beta = (\Delta\beta_1, \cdots, \Delta\beta_n)$ as depicted in Fig.3.

Then the probability of this differential is like that;

$$DP(\Delta\alpha \to \Delta\beta)$$
$$= \sum_{\Delta\delta_1,\cdots,\Delta\delta_n} \left(\prod_{i=1}^{n} DP^{S_i}(\Delta\alpha_i \to \Delta\delta_i) \prod_{i=1}^{n} DP^{S_i}(\Delta\gamma_i \to \Delta\beta_i | \Delta\alpha) \right) \tag{4}$$

Lemma 2. *Let M be the $n \times n$ matrix representing a diffusion layer D and $n_d(D) = n + 1$. In Fig.3, if $Hw(\Delta\alpha) = k$ and $Hw(\Delta\beta) = n - s + 1 (s \leq k)$, there is a index set $\{i_1, \cdots, i_{s-1}\}$ so that $\Delta\alpha_{i_1} \neq 0, \cdots, \Delta\alpha_{i_{s-1}} \neq 0$ and $\{\Delta\delta_{i_1}, \cdots, \Delta\delta_{i_{s-1}}\}$ are determined by the other $\Delta\delta_i$'s.*

Note Since $n_d(D) = n + 1$, s must be less than or equal to k. A index set $\{i_1, \cdots, i_{s-1}\}$ depends on the location of the nonzero $\Delta\alpha$ and $\Delta\beta$.
Proof Without loss of generality we may assume

$$\Delta\beta_1 = 0, \cdots, \Delta\beta_{s-1} = 0 \text{ (or equivalently} \Delta\gamma_1 = 0, \cdots, \Delta\gamma_{s-1} = 0).$$

Let $\Delta\delta' = (\Delta\delta_{i_1} \cdots, \Delta\delta_{i_k})^t$ be the collection of all non-zero components in $\Delta\delta = (\Delta\delta_1, \cdots, \Delta\delta_n)^t$. That is, $\Delta\delta_{i_j} \neq 0$ for all $1 \leq j \leq k$ and $\Delta\delta_t = 0$ if $t \notin \{i_1, \cdots, i_k\}$. Let

$$M' = \begin{pmatrix} m_{1i_1} & \cdots & m_{1i_{s-1}} & m_{1i_s} & \cdots & m_{1i_k} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ m_{s-1i_1} & \cdots & m_{s-1i_{s-1}} & m_{s-1i_s} & \cdots & m_{s-1i_k} \end{pmatrix}.$$

By the definitions of M' and $\Delta\delta'$ and the assumption on $\Delta\beta$, $M'\Delta\delta'$ equals 0. Let's divide $\Delta\delta'$ into two parts, $\Delta\delta_I$ and $\Delta\delta_{II}$, and M' into M_I and M_{II} as followings;

$$\Delta\delta_I = (\Delta\delta_{i_1} \cdots, \Delta\delta_{i_{s-1}})^t,\ \Delta\delta_{II} = (\Delta\delta_{i_s} \cdots, \Delta\delta_{i_k})^t,$$

$$M_I = \begin{pmatrix} m_{1i_1} & \cdots & m_{1i_{s-1}} \\ \vdots & \ddots & \vdots \\ m_{s-1i_1} & \cdots & m_{s-1i_{s-1}} \end{pmatrix} \text{ and } M_{II} = \begin{pmatrix} m_{1i_s} & \cdots & m_{1i_k} \\ \vdots & \ddots & \vdots \\ m_{s-1i_s} & \cdots & m_{s-1i_k} \end{pmatrix}.$$

From $M'\Delta\delta' = 0$, we get the equation

$$M_I\Delta\delta_I + M_{II}\Delta\delta_{II} = 0 (\text{or equivalently } M_I\Delta\delta_I = M_{II}\Delta\delta_{II}).$$

Since M_I is an invertible matrix by lemma 1, we have the equation

$$\Delta\delta_I = M_I^{-1}M_{II}\delta_{II}.$$

Hence $\{\Delta\delta_{i_1} \cdots, \Delta\delta_{i_{s-1}}\}$ are determined by $\{\Delta\delta_{i_s} \cdots, \Delta\delta_{i_k}\}$ ■

Lemma 2 means that the summation in (4) is not taken for all $\Delta\delta_{i_1}, \cdots, \Delta\delta_{i_k}$ but taken for all $\Delta\delta_{j_1}, \cdots, \Delta\delta_{j_{k-s+1}}$ for some index set $\{j_1, \cdots j_{k-s+1}\} \subset \{i_1, \cdots, i_k\}$. Now, we are ready to prove our main theorem.

Theorem 1. *Assume that the round keys, which are xored to the input data at each round, are independent and uniformly random. If $n_d = n+1$, the probability of each differential of SDS function is bounded by p^n.*

Proof Consider the differential as depicted in Fig.3. Let $Hw(\Delta\alpha) = k$ and $Hw(\Delta\beta) = n - s + 1$ $(s \leq k)$, then without loss of generality we may assume

$$\Delta\alpha_1 \neq 0, \cdots, \Delta\alpha_k \neq 0 \tag{5}$$

(equivalently, $\Delta\delta_1 \neq 0, \cdots, \Delta\delta_k \neq 0$) and

$$\Delta\beta_{j_1} \neq 0, \cdots, \Delta\beta_{j_{n-s+1}} \neq 0. \tag{6}$$

(equivalently, $\Delta\gamma_{j_1} \neq 0, \cdots, \Delta\gamma_{j_{n-s+1}} \neq 0$). Then,

$$\begin{aligned} & DP(\Delta\alpha \to \Delta\beta) \\ &= \sum_{\Delta\delta_1,\cdots,\Delta\delta_n} \left(\prod_{i=1}^{n} DP^{S_i}(\Delta\alpha_i \to \Delta\delta_i) \prod_{i=1}^{n} DP^{S_i}(\Delta\gamma_i \to \Delta\beta_i|\Delta\alpha) \right) \\ &= \sum_{\Delta\delta_1,\cdots,\Delta\delta_n} \left(\prod_{i=1}^{n} DP^{S_i}(\Delta\alpha_i \to \Delta\delta_i) \prod_{i=1}^{n} DP^{S_i}(\Delta\gamma_i \to \Delta\beta_i) \right) & (7) \\ &= \sum_{\Delta\delta_1,\cdots,\Delta\delta_k} \left(\prod_{i=1}^{k} DP^{S_i}(\Delta\alpha_i \to \Delta\delta_i) \prod_{i=1}^{n-s+1} DP^{S_{j_i}}(\Delta\gamma_{j_i} \to \Delta\beta_{j_i}) \right) & (8) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{\Delta\delta_{i_1}\neq 0,\cdots,\Delta\delta_{i_{k-s+1}}\neq 0} \left(\prod_{i=1}^{k} DP^{S_i}(\Delta\alpha_i \to \Delta\delta_i) \prod_{i=1}^{n-s+1} DP^{S_{j_i}}(\Delta\gamma_{j_i} \to \Delta\beta_{j_i}) \right) && (9)\\
&\le \sum_{\Delta\delta_{i_1}\neq 0,\cdots,\Delta\delta_{i_{k-s+1}}\neq 0} \left(\prod_{t=1}^{k-s+1} DP^{S_{i_t}}(\Delta\alpha_{i_t} \to \Delta\delta_{i_t}) p^{s-1} p^{n-s+1} \right) && (10)\\
&= p^n \sum_{\Delta\delta_{i_1}\neq 0,\cdots,\Delta\delta_{i_{k-s+1}}\neq 0} \left(\prod_{t=1}^{k-s+1} DP^{S_{i_t}}(\Delta\alpha_{i_t} \to \Delta\delta_{i_t}) \right)\\
&\le p^n
\end{aligned}
$$

Equation (7) follows from the assumption on round keys; equation (8) follows from assumptions (5) and (6); equation (9) follows from lemma 2; and equation (10) follows from the definition of p. ■

This theorem gives a provable security for the SPN structure.

For example, consider a 128-bit SPN structure with 16 substitution boxes, $S_1, \cdots S_{16}$, and a maximal diffusion layer. If we let

$$
\begin{array}{rcl}
S_i & : & GF(2^8) \to GF(2^8) \ (1 \le i \le n) \\
 & & \quad x \quad \to \quad x^{-1}
\end{array}
$$

we can take $p = 2^{-6}$, so that the maximum differential probability of this SDS function is bounded by $p^{16} = (2^{-6})^{16} = 2^{-96}$. Hence one gets a SPN structure which gives proven resistance of order 2^{-96} against DC.

4 Provable Security against LC

In this section we will give a provable security for the SPN structure with a maximal diffusion layer against LC. We know that the rank of M equals that of M^t for any matrix M and so applying lemma 1 and 2 gives the following result; If $n_d(D)$ is equal to $n+1$, $n_l(D)$ is also $n+1$ and vice versa. Therefore we have the following theorem.

Theorem 2. *If $n_l(D) = n+1$(or equivalently $n_d(D) = n+1$), the probability of each linear hull of SDS function is bounded by q^n.*

5 Provable Security against DC and LC with a Semi-maximal Diffusion Layer

In this section we will show that the probability of each differential is bounded by p^{n-1} when $n_d(D)$ is equal to n. A diffusion layer is called semi-maximal with respect to DC(resp. LC) when $n_d(D)$(resp. $n_l(D)$) equals n. In general $n_d(D)$ is not equal to $n_l(D)$ but there are sufficient conditions that $n_l(D)$ is equal to $n_d(D)$[14]. A diffusion layer is called **semi – maximal** if $n_d(D)$ and $n_l(D)$ are equal to n.

Lemma 3. *If $n_d(D) = n$, the rank of each $k \times k$ submatrix of M is greater than or equal to $k-1$ for all $1 \leq k \leq n$ and there exists at least one $s \times s$ submatrix with rank $s-1$ for some $1 \leq s \leq n$.*

Proof Let $n_d(D) = n$ and suppose that there exists a $k \times k$ submatrix M_k of M whose rank is less than $k-1$. That is, there exist at least two independent vectors $v, w \in GF(2^m)^k$ so that $M_k v = M_k w = 0$. We can make a vector $x \in GF(2^m)^k$ with $Hw(x) \leq k-1$ and $M_k x = 0$ by a linear combination of v and w over $GF(2^m)$. From x and M_k we can get a vector $X \in GF(2^m)^n$ such that $Hw(X) \leq k-1$ and $Hw(MX) \leq n-k$. This is contradiction to the fact that $n_d(D)$ is equal to n. Hence the rank of each $k \times k$ submatrix of M is greater than or equal to $k-1$ for all $1 \leq k \leq n$. By lemma 1 there exists at least one $s \times s$ submatrix with rank $s-1$. ■

We also state a statement similar to lemma 2; Let M be the $n \times n$ matrix representing a diffusion layer D and $n_d(D) = n$. In Fig.3, if $Hw(\Delta\alpha) = k$ and $Hw(\Delta\beta) = n - s (s \leq k)$, there is a index set $\{i_i, \cdots, i_{s-1}\}$ so that $\{\Delta\delta_{i_1}, \cdots, \Delta\delta_{i_{s-1}}\}$ are represented by the other $\Delta\delta_i$'s. The proof of this statement is similar to that of the lemma 2.

Theorem 3. *Assume that the round keys, which are xored to the input data at each round, are independent and uniformly random. If $n_d = n$, the probability of each differential of SDS function is bounded by p^{n-1}.*

Proof We use the same notations as used in the proof of theorem 1. There is only one difference between the proof of theorem 3 and that of this theorem; $Hw(\Delta\beta)$ is not $n-s+1$ but $n-s$. Thus $DP(\Delta\alpha \to \Delta\beta)$ goes up by p^{-1}. Hence we have

$$DP(\Delta\alpha \to \Delta\beta) \leq p^{n-1} \quad ■$$

We can easily check that if $n_l(D) = n$, the probability of linear hull of SDS function is bounded by q^{n-1}.

6 Conclusion

In the SPN structure, it is very important to design a diffusion layer with good properties as well as a substitution layer. Even though a substitution layer is strong against DC and LC, this does not guarantee a secure SPN structure against DC and LC if a diffusion layer does not provide an avalanche effect, both in the context of differences and linear approximations.

In this paper we give the necessary and sufficient condition for diffusion layer to be maximal or semi-maximal. Also we proved that the probability of each differential(resp. linear hull) of the SDS function with a maximal diffusion layer is bounded by p^n(resp. q^n) and that of each differential(resp. linear hull) of the SDS function with a semi-maximal diffusion layer is bounded by p^{n-1}(resp. q^{n-1}). These results give a provable security for the SPN structure against DC and LC with a maximal diffusion layer or a semi-maximal diffusion layer. Therefore we expect to obtain a SPN structure with a higher resistance against DC and LC and a smaller number of rounds.

References

1. E. Biham and A. Shamir, *Differential Cryptanalysis of DES-like Cryptosystem*, Journal of Cryptoloy, Vol.4, pp. 3-72, 1991.
2. E. Biham and A. Shamir, *Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer*, Advanced in cryptology-CRYPTO'91, pp. 156-171, Springer-Verlag, 1991.
3. E. Biham, *On Matsui's Linear Cryptanalysis*, Advanced in cryptology-EUROCRYPT'94, pp. 341-355, Springer-Verlag, 1994.
4. J. Daemen, R. Govaerts and J. Vandewalle, *Correlation Matrices*, Proceedings of the first international workshop of the Fast Software Encryption, LNCS 1008, pp. 275-285, Springer-Verlag, 1994.
5. M. Kanda, Y. Takashima, T. Matsumoto, K. Aoki and K. Ohta, *A Strategy for Constructing Fast Functions with Practical Security against Differential and Linear Cryptanalysis*, Proceedings of SAC'98, 1998.
6. M. Matsui, *Linear cryptanalysis method for DES cipher*, Advanced in cryptology-EUROCRYPT' 93, pp. 386-397, Springer-Verlag, 1993.
7. M. Matsui, *The first Experimental cryptanalysis of DES*, Advanced in cryptology-CRYPTO'94, pp. 1-11, Springer-Verlag, 1994.
8. M. Matsui, *New Block Encryption Algorithm MISTY*, Proceedings of the fourth international workshop of Fast Software Encryption, Springer-Verlag, pp. 53-67, 1997.
9. K. Nyberg and L. R. Knudsen, *Provable security against a differential attack*, Advanced in cryptology-CRYPTO'92, pp. 566-574, Springer-Verlag, 1992.
10. K. Nyberg, *Differentially uniform mappings for cryptography*, Advanced in cryptology-EUROCRYPT'93, pp. 55-64, Springer-Verlag, 1993.
11. K. Nyberg, *Linear Approximation of block ciphers*, Advanced in cryptology-EUROCRYPT'94, pp. 439-444, Springer-Verlag, 1994.
12. V. Rijmen, J.Daemen et al, *The cipher SHARK*, Proceedings of the fourth international workshop of Fast Software Encryption, pp. 137-151, Springer-Verlag, 1997.
13. J. Daemen and V. Rijmen, *The Rijdael block cipher*, AES proposal, 1998.
14. J. Kang, C. Park, S. Lee and J. Lim, *On the optimal diffusion layer with practical security against Differential and Linear Cryptanalysis*, Preproceedings of ICISC'99, pp. 13-20, 1999.
15. X. Lai, J. L. Massey and S. Murphy *Markov Ciphers and Differential Cryptanalysis*, Advances in Cryptology-EUROCRYPT'91, pp 17-38, Springer-Verlag, 1992.
16. J. Daemen, *Cipher and hash function design strategies based on linear and differential cryptanalysis,* Doctoral Dissertation, March 1995, K.U. Leuven.
17. K. Aoki and K. Ohta, *Strict Evaluation of the Maximum Average of Differential Probability and the Maximum Average of Linear Probability,* IEICE Transactions Fundamentals of Electronics, Communications and Computer Science, Vol. E80A, No. 1, pp. 2-8, 1997.

Unforgeable Encryption and Chosen Ciphertext Secure Modes of Operation

Jonathan Katz[1] and Moti Yung[2]

[1] Department of Computer Science, Columbia University
jkatz@cs.columbia.edu
[2] CertCo, NY, USA
moti@cs.columbia.edu, moti@certco.com

Abstract. We find certain neglected issues in the study of private-key encryption schemes. For one, private-key encryption is generally held to the same standard of security as public-key encryption (i.e., indistinguishability) even though usage of the two is very different. Secondly, though the importance of secure encryption of single blocks is well known, the security of modes of encryption (used to encrypt multiple blocks) is often ignored. With this in mind, we present definitions of a new notion of security for private-key encryption called *encryption unforgeability* which captures an adversary's inability to generate valid ciphertexts. We show applications of this definition to authentication protocols and adaptive chosen ciphertext security.
Additionally, we present and analyze a new mode of encryption, RPC (for Related Plaintext Chaining), which is unforgeable in the strongest sense of the above definition. This gives the first mode provably secure against chosen ciphertext attacks. Although RPC is slightly less efficient than, say, CBC mode (requiring about 33% more block cipher applications and having ciphertext expansion of the same amount when using a block cipher with 128-bit blocksize), it has highly parallelizable encryption and decryption operations.

1 Introduction

1.1 Motivation

Much work has been devoted to developing precise definitions of security for encryption schemes [2,3,16] and to constructing cryptosystems meeting these enhanced notions of security. Currently, the same notions of security are used to analyze both public-key and private-key encryption. In the public-key setting, however, encryption is available to everyone; in this case, therefore, one need only worry about the possibility of an adversary decrypting an encrypted message. This is in contrast to the private-key setting where one must also worry about the potential harm an adversary can cause by generating the encryption of some message (an action which a protocol designer may not expect to occur). So, while one can "borrow" security notions from the public-key to the private-key setting, one has to recognize that the security goals of the latter may be different.

Additionally, private-key cryptography is used to transmit large amounts of data (in particular, more than one block at a time using some mode of operation), while public-key cryptography is generally used to send short messages (typically, session keys). Finally, private-key encryption is the basis of many authentication and security handshake protocols [14].

For these reasons, we introduce here a higher standard of security for private-key encryption called *encryption unforgeability* (simply *unforgeability* in the sequel), which characterizes an adversary's inability to generate valid ciphertexts. This notion turns out to be quite useful: it guarantees security for certain authentication protocols, provides message integrity without additional computation or cryptographic primitives, and offers some level of security under chosen ciphertext attacks.

Unforgeability is particularly interesting in the context of modes of encryption. The encryption of single blocks, both in theory and in practice, is well understood; however, we believe that the security of modes of encryption has been (comparatively) neglected. To remedy this, we present a new mode of encryption which is unforgeable in the strongest sense of the definition. We then show, using a concrete security analysis [2], that this mode is secure against the strongest form of chosen plaintext/ciphertext attacks, and is non-malleable as well.

1.2 Applicability

Unforgeability seeks to capture the following intuition: an adversary, after intercepting various ciphertext messages, should not be able to generate a new ciphertext corresponding to any valid plaintext. As an application of this, say Alice and Bob carry out a handshaking protocol using a shared private key K. Alice sends Bob $\mathcal{E}_K(\mathsf{timestamp})$, and Bob must reply with $\mathcal{E}_K(\mathsf{timestamp}+1)$ (this is similar to the protocol implemented in Kerberos V4 [14]). This is meant to prove to Alice that Bob knows the secret key K. However, if an adversary can somehow "forge" an encryption of $\mathsf{timestamp}+1$, he can authenticate himself to Alice without knowledge of the key. Note that this differs from a non-malleability attack; in the case of non-malleability, the adversary does not know the plaintext corresponding to the "challenge" ciphertext. In this case, however, the current time is known to all participants, so Bob *does* know the plaintext corresponding to Alice's encrypted message.

Extending this further, if Alice and Bob are communicating over an insecure channel by encrypting messages using a shared key, it is clearly undesirable for an adversary to be able to insert ciphertext into the channel which will be decrypted by one of the parties and interpreted as a valid (potentially malicious) message. This threat can be reduced by using message authentication, but most schemes for integrating encryption and authentication are either inefficient or are not provably secure. In fact, efficient integration of encryption, authentication, and message integrity *using a single shared key* is an important and intensely studied problem in network security [14,23] which is solved by the use of an unforgeable encryption scheme.

1.3 Importance of a Single-Key Solution

A trivial solution to the problem of unforgeability is to share two keys—one for encryption and one for authentication (e.g., using a MAC). Also, various constructions (requiring multiple shared keys) are known to exist which allow for chosen-ciphertext secure encryption of large (i.e., many block) messages [20,5,8]. However, we have in mind applications such as integrated public-key/private-key encryption (e.g., PGP for e-mail encryption) in which it is advantageous to share only one key. One public-key encryption of a session key followed by (slow) private-key encryption of the message is still faster than two public-key encryptions and (fast) private-key encryption, for "short" messages. Hardware implementations of encryption may also benefit from the single-key requirement. Furthermore, it may be required to integrate a new mode of encryption into existing software which already calls for encryption of only one session key.

1.4 Previous Work

Notions of Security. Beginning with the paper by Goldwasser and Micali [10], which first provided a rigorous definition of "semantic security", the cryptography community has progressed to the currently accepted definitions of indistinguishability (polynomially equivalent to semantic security) and non-malleability (introduced by Dolev, Dwork, and Naor [9]; later reformulated by Bellare, et al. [3]). Indistinguishability describes an adversary's inability to derive any information from a ciphertext about the corresponding plaintext. Non-malleability characterizes an adversary's inability, given access to a challenge ciphertext, to generate a different ciphertext meaningfully related to the challenge ciphertext. We refer the reader elsewhere [3,16] for formal definitions.

Unforgeability. Previous work dealing with concurrent encryption plus message authentication implicitly uses many of the ideas of unforgeability [13,23]. However, the formal definition presented here is new.

Unforgeability is distinct from non-malleability. In the latter, the adversary's goal is to generate one ciphertext meaningfully related to another. In the former (depending on the type of attack, see below), the adversary may succeed by generating *any* valid ciphertext. Furthermore, in a non-malleability-based attack, the adversary does not know the plaintext corresponding to the challenge ciphertext. This is in contrast to an unforgeability-based attack, in which queries are made to an encryption oracle and therefore the adversary may know the underlying decryption of some ciphertexts. Thus, the level of security considered here is much stronger.

Chosen Ciphertext Security. Chosen ciphertext [21] and adaptive chosen ciphertext [25] attacks are very powerful attacks in which the adversary can obtain decryptions of her choice (in the case of adaptive attacks, even after seeing the challenge ciphertext). As it is not our intention to survey the literature on chosen ciphertext security, we merely point out that most research thus far has focused on public-key encryption. Little attention has been paid to chosen ciphertext security for private-key encryption (an exception is [9]), and even less to chosen ciphertext secure modes of encryption.

Modes of Encryption. The commonly-used modes of encryption include those defined as part of the DES [1,11,22], an XOR mode suggested by Bellare, et al. [2], and the PCBC mode [19] used in Kerberos V4. The security of these modes of encryption lags behind the security of available block encryption algorithms. None of the above modes are non-malleable, and all are vulnerable to an adaptive chosen ciphertext attack. This has been recognized in the cryptographic literature for some time [19], but the previously-mentioned modes continue to be used even though potentially serious security flaws may result [15].

Security analyses of modes of encryption have focused on chosen plaintext attacks. Examples include a concrete analysis of the CBC and XOR modes [2], Biham's and Knudsen's analyses of modes of operation of DES and triple-DES [6,7], and others [13,24].

A mode of encryption as secure as a pseudorandom permutation is given by Bellare and Rogaway [5]. The only other examples of (potentially) chosen-ciphertext-secure modes of encryption of which we are aware [20,8] study a slightly different problem, and are therefore not *provably* chosen-ciphertext secure. Our suggestion (RPC) lends itself to simpler analysis and has certain practical advantages over these other constructions; we refer the reader to the Discussion in Section 5.

1.5 Summary of Results

We begin in Section 2 with a review of some notation. In Section 3 we present definitions for three levels of unforgeability for private-key encryption; this section concludes with a theorem formalizing the relation between security in the sense of unforgeability and security under chosen ciphertext attacks. Section 4 describes two simple modes of encryption which are unforgeable under the strongest definition. We conclude with a discussion of the practicality of these modes, and a comparison with previously suggested modes which are potentially chosen ciphertext secure.

The unforgeable modes we present are actually quite straightforward. The intuition is as follows: to prevent an adversary from "splicing" together blocks from different ciphertexts, we "tag" each ciphertext block with a sequence number. To prevent an adversary from shortening a ciphertext to generate a new, valid one, we "mark" the beginning and end of each ciphertext with special start and end tokens. Encryption of $x_1, \ldots, x_n$ is simply given by:

$$\mathcal{E}(\mathsf{start}, i), \mathcal{E}(x_1, i+1), \ldots, \mathcal{E}(x_n, i+n), \mathcal{E}(\mathsf{end}, i+n+1),$$

where the nature of i depends on the details of the mode.

It follows from the properties of RPC that it can provide a single-key, one-pass, provably secure mechanism for "concurrent encryption, authentication, and message integrity" using a single shared key (an open question [14,23]). The encryption and decryption operations are parallelizable, and the mode works better with larger blocksizes (i.e., it is better suited for AES than for DES). Because of these properties and its simplicity, we suggest RPC as a practical addition to (or substitute for) the modes of encryption currently in use.

2 Notation

Indistinguishability. We follow the standard notation for private-key encryption [16], but note that we consider both probabilistic and stateful encryption schemes. We use the notion of indistinguishability as our measure of security. The definition below is similar to those given elsewhere [2,16]; we rephrase it to allow for a concrete security analysis. Also, since we deal explicitly with the security of modes of encryption, we parameterize the adversary's advantage by the length of the submitted plaintexts. The notation for notions of security follows [16]; thus, IND-PX-CY means an indistinguishability-based attack, with encryption oracle access at level X and decryption oracle access at level Y. Level 0 denotes no oracle access, level 1 denotes access before being presented with the challenge ciphertext (non-adaptive access), and level 2 denotes access both before *and after* the challenge ciphertext is revealed (adaptive access). Of course, different levels of access can be chosen separately for each oracle. The definition below corresponds to security under an IND-P2-C2 attack.

Definition 1. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme accepting variable length messages, and let $A = (A_1, A_2)$ be an adversary. Let* $\mathsf{Adv}_{A,\Pi}^{\mathrm{IND-P2-C2}} \stackrel{\mathrm{def}}{=}$

$$2 \cdot \Pr\Big[sk \leftarrow \mathcal{K}; (x_0, x_1, s) \leftarrow A_1^{\mathcal{E}_{sk}(\cdot), \mathcal{D}_{sk}(\cdot)}; b \leftarrow \{0,1\}; y \leftarrow \mathcal{E}_{sk}(x_b) : \\ A_2^{\mathcal{E}_{sk}(\cdot), \mathcal{D}_{sk}(\cdot)}(x_0, x_1, s, y) = b\Big] - 1.$$

We insist that $|x_0| = |x_1| = \ell$. Furthermore, all queries to the encryption oracle consist of an integral number of blocks (the size of which depends on the underlying block cipher algorithm); thus, no "padding" is ever required.

We say that Π is $(t, q_e, b_e, q_d; \ell, \epsilon)$-secure in the sense of IND-P2-C2 if for any adversary A distinguishing between plaintexts of length ℓ which runs in time at most t, submits at most q_e queries to the encryption oracle (these totaling at most b_e bits), and submits at most q_d queries to the decryption oracle we have $\mathsf{Adv}_{A,\Pi}^{\mathrm{IND-P2-C2}} \leq \epsilon$.

Note that IND-P2-C2 security implies security under all other notions of indistinguishability and non-malleability [16].

Super Pseudorandom Permutations (following [2]). A *permutation family* is a set F of permutations all having the same domain and range. We assume the permutations are indexed by some key $k \in K$, such that F_k specifies a particular permutation in F. Usually, K is the set of all strings of some fixed length. We write $f \leftarrow F$ to denote selection of a permutation at random from F according to the distribution given by picking a random k and setting $f = F_k$. Let P_n be the permutation family consisting of all permutations on $\{0,1\}^n$. Thus, $f \leftarrow P_n$ means selection of a random permutation on n-bit strings.

Let F, G be function families with the same domain and range. Consider a *distinguisher* D, given oracle access to a function f and its inverse f^{-1}, that attempts to distinguish between the case where f is chosen randomly from F and the case where f is chosen randomly from G. Let

$$\mathsf{Dist}_D(F,G) = \Pr\left[f \leftarrow F : D^{f(\cdot),f^{-1}(\cdot)} = 1\right] - \Pr\left[f \leftarrow G : D^{f(\cdot),f^{-1}(\cdot)} = 1\right].$$

A super pseudorandom permutation (super-PRP) [18] family F on $\{0,1\}^n$ has the property that the input-output behavior of f, f^{-1} "looks random" to someone who does not know the randomly selected key k. Accordingly, define the advantage of the distinguisher by:

$$\mathsf{Adv}_D^{\mathsf{SPRP}}(F) = \mathsf{Dist}_D(F, P_n).$$

Definition 2. *We say that super-PRP family F is $(t, q_1, q_2; \epsilon)$-secure if for any distinguisher D which makes at most q_1 oracle queries of f, q_2 oracle queries of f^{-1}, and runs in time at most t it is the case that $\mathsf{Adv}_D^{\mathsf{SPRP}}(F) \leq \epsilon$.*

Note the distinction from a pseudorandom permutation (PRP); in the latter case, the distinguisher is only given access to the function f (not its inverse f^{-1}). However, any family of PRPs can be converted to a family of super-PRPs [18].

3 Unforgeability

3.1 Definitions

We define three levels of unforgeability for encryption, progressing from the weakest to the strongest. In the following definitions, we assume that the set of valid encryption oracle queries is exactly the same as the valid message space; this eliminates technical problems arising from having to randomly pad short messages. Note that in all cases our definitions do not restrict the length of the forged ciphertext or the length of the plaintext queries submitted to the encryption oracle. Thus, our definitions include cases where an adversary might try to extend previous ciphertexts, paste two ciphertexts together, or delete blocks from a valid ciphertext in an attempt to forge a message.

Our definitions focus on the settings with maximum access to an encryption oracle. Weaker definitions, with non-adaptive access or no access, are also possible.

Random Plaintext Unforgeability. The framework of the attack is as follows: a challenge plaintext x is chosen at random from the message space M. The adversary succeeds if he can output a ciphertext y such that x is the decryption of y. This essentially means that the adversary has "broken" one direction of the encryption, since he can forge ciphertext for just about any message he chooses.

Definition 3. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a private-key encryption scheme, and let A be an adversary. Define:*

$$\mathsf{Adv}^{\text{random}}_{A,\Pi} \stackrel{\text{def}}{=} \Pr\left[sk \leftarrow \mathcal{K}; x \leftarrow M; y \leftarrow A^{\mathcal{E}_{sk}(\cdot)}(x) : \mathcal{D}_{sk}(y) = x\right].$$

We insist that A does not ask the oracle to encrypt x. We say that Π is $(t, q, b; \epsilon)$-secure in the sense of random plaintext unforgeability if for any adversary A which runs in time at most t and asks at most q queries, these totaling at most b bits, we have $\mathsf{Adv}^{\text{random}}_{A,\Pi} \leq \epsilon$.

Chosen Plaintext Unforgeability. In this attack, the goal of the adversary is simpler. Instead of having to find the encryption of a "challenge" plaintext, the adversary is free to output the encryption of any plaintext he chooses. But, the adversary must know the plaintext message to which this ciphertext corresponds. This is similar to the *valid pair creation* attack defined previously [12].

Definition 4. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a private-key encryption scheme, and let A be an adversary. Define:*

$$\mathsf{Adv}^{\text{chosen}}_{A,\Pi} \stackrel{\text{def}}{=} \Pr\left[sk \leftarrow \mathcal{K}; (x, y) \leftarrow A^{\mathcal{E}_{sk}(\cdot)} : \mathcal{D}_{sk}(y) = x\right].$$

We insist, above, that A has never received ciphertext y in return from its encryption oracle. We say that Π is $(t, q, b; \epsilon)$-secure in the sense of chosen plaintext unforgeability if for any adversary A which runs in time at most t and asks at most q queries, these totaling at most b bits, we have $\mathsf{Adv}^{\text{chosen}}_{A,\Pi} \leq \epsilon$.

Existential Unforgeability. This represents the strongest notion of unforgeability, as it corresponds to the simplest attack for an adversary. The adversary succeeds by producing *any* new valid ciphertext, even without knowing the corresponding plaintext.

Definition 5. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a private-key encryption scheme, and let A be an adversary. Define:*

$$\mathsf{Adv}^{\text{exist}}_{A,\Pi} \stackrel{\text{def}}{=} \Pr\left[sk \leftarrow \mathcal{K}; y \leftarrow A^{\mathcal{E}_{sk}(\cdot)} : \mathcal{D}_{sk}(y) \neq \perp\right].$$

We insist, above, that A has never received ciphertext y in return from its encryption oracle. We say that Π is $(t, q, b; \epsilon)$-secure in the sense of existential unforgeability if for any adversary A which runs in time at most t and asks at most q queries, these totaling at most b bits, we have $\mathsf{Adv}^{\text{exist}}_{A,\Pi} \leq \epsilon$.

3.2 Unforgeability and Chosen Ciphertext Security

The notion of existential unforgeability is a strong one; as such, we expect some relation between existential unforgeability and chosen ciphertext security. The intuition is clear: since any (new) ciphertexts generated by an adversary are likely to be invalid, the adversary cannot gain much by submitting them to the decryption oracle. Thus, if a scheme cannot be broken with no access to a decryption oracle, it cannot be broken much more often even when access to a decryption oracle is given. This is formalized by the following theorem, which shows that unforgeability (along with chosen plaintext security) implies adaptive chosen plaintext/ciphertext security.

Theorem 1. *Let Π be an encryption scheme which is $(t_1, q, b; \epsilon_1)$-secure in the sense of existential unforgeability, and $(t_2, q_e, b_e; \ell, \epsilon_2)$-secure in the sense of IND-PX-C0 (for $X \in \{0,1,2\}$). Then Π is $(t', q'_e, b'_e, q_d; \ell, \epsilon')$-secure in the sense of IND-PX-C2, where $t' = \min\{t_1, t_2\}$; $q'_e = \min\{q-1, q_e\}$; $b'_e = \min\{b-\ell, b_e\}$; and $\epsilon' = \epsilon_2 + q_d\epsilon_1$.*

Sketch of Proof Say adversary A attacks Π in the sense of IND-PX-C2, running in time t', making at most q'_e encryption oracle queries totaling at most b'_e bits, and making q_d decryption oracle queries. We assume without loss of generality that A never queries the decryption oracle on a ciphertext which it received in return from the encryption oracle (in fact, there is no reason for A to do so). We can construct the following adversaries:

1. Adversaries $\{B_i\}$ (for $i = 1, \ldots, q_d$) attacking Π in the sense of existential unforgeability. B_i runs A as a subroutine, and answers the first $i-1$ decryption oracle queries made by A with $\perp$, then returns as its output the i^{th} decryption oracle query made by A.
2. Adversary C attacking Π in the sense of IND-PX-C0, which runs A as a subroutine but answers all decryption oracle queries made by A with $\perp$.

Define Valid_i to be the event that the i^{th} decryption oracle query submitted by A was the first one to be valid. Extending this notation, let Valid_∞ be the event that none of the decryption oracle queries submitted by A are valid. Let Succ be (informally) the event that A succeeds in distinguishing the challenge ciphertext it is given. We have:

$$\begin{aligned}
\mathsf{Adv}_{A,\Pi}^{\mathrm{IND-PX-C2}} &= \\
&\Pr\left[\mathsf{Succ}|\mathsf{Valid}_{q_d} \vee \cdots \vee \mathsf{Valid}_1\right] \Pr\left[\mathsf{Valid}_{q_d} \vee \cdots \vee \mathsf{Valid}_1\right] \\
&\quad + \Pr\left[\mathsf{Succ}|\mathsf{Valid}_\infty\right] \Pr\left[\mathsf{Valid}_\infty\right] \\
&\le \sum_{i=1}^{q_d} \Pr\left[\mathsf{Valid}_i\right] + \Pr\left[\mathsf{Succ}|\mathsf{Valid}_\infty\right] \Pr\left[\mathsf{Valid}_\infty\right] \\
&\le \sum_{i=1}^{q_d} \mathsf{Adv}_{B_i,\Pi}^{\mathrm{exist}} + \mathsf{Adv}_{C,\Pi}^{\mathrm{IND-PX-C0}}.
\end{aligned}$$

Since the advantages of adversaries $\{B_i\}$ and C are bounded by ϵ_1 and ϵ_2 respectively, we have:

$$\mathsf{Adv}_{A,\Pi}^{\mathrm{IND-PX-C2}} \leq q_d\epsilon_1 + \epsilon_2.$$

This gives the stated result. □

3.3 The Forgeability of Previous Modes

It is instructive to demonstrate attacks on the original DES modes of encryption [1,11,22]. It is clear that all of these modes are susceptible to an existential unforgeability attack (even a passive attack, with no encryption oracle access), since any ciphertext string of appropriate length decrypts to a valid plaintext. Some modes are even weaker. ECB mode is susceptible to a random plaintext unforgeability attack (with adaptive encryption oracle access) as follows: to find the encryption of $M_1, M_2, \ldots, M_l$ simply submit to the encryption oracle the two queries M_1 and $M_2, \ldots, M_l$ and paste the responses together. OFB has even worse characteristics—it is susceptible to a random plaintext unforgeability attack with *non-adaptive* encryption oracle access. Simply have the adversary submit $0^{n\ell}$ to the encryption oracle and receive ciphertext $C_0, C_1, \ldots, C_\ell$. To forge encryption of $M_1, \ldots, M_\ell$, compute $C_i' = C_i \oplus M_i$ and return $C_0, C_1', \ldots, C_\ell'$. Attacks in the sense of chosen plaintext unforgeability exist for CBC and CFB modes as well [15,19]. These examples indicate the weaknesses of these modes; this further implies that these modes are not adaptive-chosen-ciphertext secure.

4 Unforgeable Modes of Operation

We present two modes of encryption (one stateful, one probabilistic) which are secure under the strongest definition of unforgeability and are additionally secure under adaptive chosen plaintext/ciphertext attack. The encryption (decryption) algorithms use underlying block cipher (using secret key *sk*) F_{sk}. The mode is parameterized by n and r, where the underlying block cipher operates on n-bit blocks and r is the length of the padding.

4.1 A Stateful Mode

We begin by describing the stateful mode of encryption. The variable *ctr* is an r-bit binary number; it is initialized to 0, and addition is modulo 2^r. (We assume that the start and end symbols do not represent valid message blocks.)

```
Algorithm $\mathcal{E}$-RPC$_{n,r}(ctr, M)$
  parse $M$ as $M_1, \ldots, M_\ell$,
   where $|M_i| = n - r$
  $C_0 = F_{sk}(\mathsf{start}, ctr)$
  for $i = 1, \ldots, \ell$ do
    $C_i = F_{sk}(M_i, ctr + i)$
  $C_{\ell+1} = F_{sk}(\mathsf{end}, ctr + \ell + 1)$
  $ctr := ctr + \ell + 1$
  return $(ctr, C = C_0, \ldots, C_{\ell+1})$
```

```
Algorithm $\mathcal{D}$-RPC$_{n,r}(C)$
  parse $C$ as $C_0, \ldots, C_{\ell+1}$,
   where $|C_i| = n$
  if $\ell + 1 < 3$ return $\perp$
  for $i = 0, \ldots, \ell + 1$ do
    $(M_i, ctr_i) = F_{sk}^{-1}(C_i)$
  if $(M_0 \neq \mathsf{start}) \vee (M_{\ell+1} \neq \mathsf{end})$ return $\perp$
  for $i = 1, \ldots, \ell$ do
    if $ctr_i \neq ctr_0 + i$ return $\perp$
    if $(M_i = \mathsf{start}) \vee (M_i = \mathsf{end})$ return $\perp$
  if $ctr_{\ell+1} \neq ctr_0 + \ell + 1$ return $\perp$
  return $M = M_1, \ldots, M_\ell$
```

Theorems 2 and 3 quantify the security of RPC mode when instantiated with a fully random permutation:

Theorem 2. *Let Π be an encryption scheme using $RPC_{n,r}$ mode instantiated with a block cipher chosen randomly from P_n. Then Π is $(t, q, b(n-r); \epsilon)$-secure in the sense of existential unforgeability (for $b + q \leq 2^r$), where:*

$$\epsilon \leq \frac{2^{n-r-1}}{2^n - b - 2q}.$$

Sketch of Proof Recall the technical assumption that the adversary submits oracle queries whose lengths are integer multiples of $n - r$ bits (i.e., no padding is required). Without this assumption, it is unclear how to define a notion of unforgeability. Thus, the adversary submits b blocks of plaintext to the oracle.

Due to the construction of the mode, the adversary will have to introduce at least one new (previously-unseen) ciphertext block in the output ciphertext. The *ctr* variable derived from this block must "match up" with the remainder of the message. There are at most 2^{n-r-1} blocks whose *ctr* will match up properly (there are $n - r$ data bits, and these cannot take on the values start or end). Furthermore, there is a pool of at least $2^n - b - 2q$ ciphertext blocks to choose from after eliminating those blocks which the adversary has already received in return from the encryption oracle (b blocks of data and $2q$ blocks to account for encryption of start and end tokens). Since we are dealing with a random permutation, this gives the stated result. □

This bound is tight, as an adversary can submit one plaintext block to its encryption oracle, receive in return ciphertext $C = C_0, C_1, C_2$, and then output $C' = C_0, C_1', C_2$ as an attempted forgery. Clearly, C' is valid iff the *ctr* variable derived from C_1' "matches up" with the remainder of the ciphertext, and this occurs with the probability specified in the theorem.

Theorem 3. *Let Π be an encryption scheme using RPC$_{n,r}$ mode instantiated with a block cipher chosen randomly from P_n. Then Π is $(t, q_e, b_e(n-r), q_d; \ell, \epsilon)$-secure in the sense of IND-P2-C2 (for $b_e + q_e + \ell + 2 \leq 2^r$), where:*

$$\epsilon = \frac{q_d 2^{n-r-1}}{2^n - b_e - 2q_e}.$$

Proof Due to the stateful mode of operation and the fact it uses a random permutation, the advantage of any adversary attacking Π in the sense of IND-P2-C0 is 0. Application of Theorem 1 gives the desired result. □

These results translate into the following "real-world" security:

Theorem 4. *Suppose F is a $(t, q_1, q_2; \epsilon)$-secure super-PRP family ($q_2 \geq 2$). Let Π be an encryption scheme using RPC$_{n,r}$ mode instantiated with a block cipher chosen randomly from F. Then Π is $(O(t - b \log b), q, b(n-r); \epsilon')$-secure in the sense of existential unforgeability (for $b + q \leq 2^r$ and $b + 2q \leq q_1$), where:*

$$\epsilon' = \epsilon + \frac{2^{n-r-1}}{2^n - b - 2q}.$$

Sketch of Proof Assume adversary A attacks Π in the sense of existential unforgeability. Without loss of generality, we may assume that A does not output a ciphertext which it has obtained in response from its encryption oracle. Construct a distinguisher D for F which will use A as a subroutine. D simulates A's encryption oracle by maintaining an internal *ctr* variable, "padding" A's oracle queries according to the definition of RPC$_{n,r}$, and submitting the resulting blocks to its own oracle for f. When A returns a (supposedly forged) ciphertext, D finds a block in this ciphertext which it did not previously receive from f and submits that block (and an adjacent block, if that too has never been received from f) to its oracle for f^{-1}. (Note that submitting the entire ciphertext is unnecessary, since our analysis in Theorem 2 bounds the probability of forging even one block.) If the *ctr* variables "match up", D outputs 1 (guessing that f was chosen from F); otherwise, D outputs 0.

This requires D to submit $b + 2q$ queries to its oracle for f, and 2 queries to its oracle for f^{-1}. Running time includes time to update the counter and to sort and search through the submissions/responses from oracle f. The theorem follows. □

Theorem 5. *Suppose F is a $(t, q_1, q_2; \epsilon)$-secure super-PRP family. Let Π be an encryption scheme using RPC$_{n,r}$ mode instantiated with a block cipher chosen randomly from F. Then Π is $(O(t - b_e \log b_e), q_e, b_e(n-r), q_d; \ell, \epsilon')$-secure in the sense of IND-P2-C2 (for $b_e + q_e + \ell + 2 \leq 2^r$ and $2q_d \leq q_2$), where:*

$$\epsilon' = \epsilon + \frac{q_d 2^{n-r-1}}{2^n - b_e - 2q_e}.$$

Sketch of Proof The proof is similar to that of Theorem 4. Construct distinguisher D from adversary A. For each ciphertext submitted by A to the decryption oracle, D finds a block in the ciphertext which was not previously received from its oracle for f. D submits that block (and an adjacent block, if that too has never been received from f) to its oracle for f^{-1}. If the *ctr* variables "match up", D outputs 1 (guessing that f was chosen from F); otherwise, D returns $\perp$ to A. D also outputs 1 if A successfully distinguishes the ciphertext even though all its decryption oracle queries were answered by $\perp$. The proof follows. □

4.2 A Probabilistic Mode

The probabilistic mode is a straightforward extension of the stateful mode. We present it here for completeness (*rand* represents an r-bit binary number, and addition is done modulo 2^r):

```
Algorithm $\mathcal{E}$-RPC\$$_{n,r}(M)$
  parse $M$ as $M_1, \ldots, M_\ell$,
    where $|M_i| = n - r$
  $rand \leftarrow \{0,1\}^r$
  $C_0 = F_{sk}(\mathsf{start}, rand)$
  for $i = 1, \ldots, \ell$ do
    $C_i = F_{sk}(M_i, rand + i)$
  $C_{\ell+1} = F_{sk}(end, rand + \ell + 1)$
  return $C = C_0, \ldots, C_{\ell+1}$
```

```
Algorithm $\mathcal{D}$-RPC\$$_{n,r}(C)$
  parse $C$ as $C_0, \ldots, C_{\ell+1}$,
    where $|C_i| = n$
  if $\ell + 1 < 3$ return $\perp$
  for $i = 0, \ldots, \ell + 1$ do
    $(M_i, rand_i) = F_{sk}^{-1}(C_i)$
  if $(M_0 \neq \mathsf{start}) \vee (M_{\ell+1} \neq \mathsf{end})$ return $\perp$
  for $i = 1, \ldots, \ell$ do
    if $rand_i \neq rand_0 + i$ return $\perp$
    if $(M_i = \mathsf{start}) \vee (M_i = \mathsf{end})$ return $\perp$
  if $rand_{\ell+1} \neq rand_0 + \ell + 1$ return $\perp$
  return $M = M_1, \ldots, M_\ell$
```

The following theorems quantify the security of RPC\$ mode with respect to existential unforgeability and chosen ciphertext attacks.

Theorem 6. *Let Π be an encryption scheme using RPC\$$_{n,r}$ mode instantiated with a block cipher chosen randomly from P_n. Then Π is $(t, q, b(n-r); \epsilon_{\text{unf}})$-secure in the sense of existential unforgeability, where:*

$$\epsilon_{\text{unf}} = \frac{(b+q)(q-1)}{2^r} + \frac{2^{n-r-1}}{2^n - b - 2q}.$$

Sketch of Proof Let A be an adversary attacking Π in the sense of existential unforgeability. Let $rand^i$ be the nonce associated with query i, for $i = 1, \ldots, q$, and let b^i be the number of plaintext blocks in the ith query. Let $\mathsf{Overlap}$ be the event that $rand^i + k = rand^j + k'$ for some $i \neq j$ and $0 \leq k \leq b^i, 0 \leq k' \leq b^j$. In other words, $\mathsf{Overlap}$ is the event that there is an overlapping sequence in the random padding used.

The success probability of A is given by:

$$\begin{aligned}\Pr[\mathsf{Success}] &= \Pr[\mathsf{Success}|\mathsf{Overlap}]\Pr[\mathsf{Overlap}] \\ &\quad + \Pr\left[\mathsf{Success}|\overline{\mathsf{Overlap}}\right]\Pr\left[\overline{\mathsf{Overlap}}\right] \\ &\leq \Pr[\mathsf{Overlap}] + \Pr\left[\mathsf{Success}|\overline{\mathsf{Overlap}}\right].\end{aligned}$$

Now, in the case of $\overline{\mathsf{Overlap}}$, the advantage of A is the same as in the stateful version of RPC (Theorem 2). Furthermore, we have (following [2]):

$$\Pr[\mathsf{Overlap}] < \frac{(b+q)(q-1)}{2^r}.$$

This gives the stated result. □

To see that this bound is essentially tight, consider an adversary A making $q = 2$ queries, totaling b blocks, achieving success probability approximately $\frac{2^{n-r-1}}{2^n-b-2q} + \frac{b-2}{2^r}$, which operates as follows: A submits to the encryption oracle the plaintext $0^{(b/2)(n-r)}$ *twice*. If any of the ciphertext blocks received in return from the oracle are equal (except in the case that the two ciphertexts received are entirely equal), A can cut-and-paste the ciphertexts to form a new ciphertext decrypting to a (longer or shorter) valid plaintext consisting of all zeros. The probability of this occurring is precisely $\frac{b-2}{2^r}$. If this does not happen, A guesses a value for a ciphertext block and submits this (as before).

Theorem 7. *Let Π be an encryption scheme using RPC\$$_{n,r}$ mode instantiated with a block cipher chosen randomly from P_n. Then Π is $(t, q_e, b_e(n-r), q_d; k, \epsilon)$-secure in the sense of IND-P2-C2, where:*

$$\epsilon = \frac{(k-1)q_e + b_e}{2^r} + q_d\epsilon_{\text{unf}}.$$

Sketch of Proof We first analyze the success probability of an adversary A when attacking Π in the sense of IND-P2-C0. Let $rand^i$ be the nonce associated with query i, for $i = 1, \ldots, q_e$, and let b^i be the number of plaintext blocks in the i^{th} query. Let $rand^*$ be the nonce associated with the challenge ciphertext. It is clear that the adversary can succeed only when $rand^i + r = rand^* + r'$, with $1 \leq r \leq b^i, 1 \leq r' \leq k$ (indeed, an adversary can submit the plaintexts $0^{k(n-r)}$, $1^{k(n-r)}$ and then query the encryption oracle repeatedly at $0^{k'(n-r)}$ for various values of k' to try for a repeated block). The chance of such a collision is maximized if all nonce sequences generated by encryption oracle queries are no less than $k-1$ blocks apart. Then, a collision occurs if $rand^*$ is $k-1$ or fewer blocks before any previous sequence or in a block occupied by some previous sequence. So:

$$\mathsf{Adv}_{A,\Pi}^{\text{IND-P2-C0}} \leq \frac{(k-1)q_e + b_e}{2^r}.$$

Application of Theorems 6 and 1 gives the final result. □

Bounds for RPC\$ instantiated with a super-PRP block cipher are straightforward extensions of the above (as in Theorems 4 and 5), and are omitted.

4.3 Forthcoming: An Incremental Mode

The present results have led us to develop a chosen-ciphertext-secure incremental encryption mode [4]; details will appear in a forthcoming manuscript [17]. An incremental mode of encryption is one in which updating the encryption of a document (for instance, when a document is edited) is much faster than re-encryption of the entire document. An incremental version of the modes presented above partially offsets their relative inefficiency (depending on the application).

5 Discussion

It is instructive to compare RPC mode to other modes of encryption which might potentially achieve security under chosen ciphertext attacks. Note, however, that previous work in this area has concentrated on extending super-PRPs on n bits to super-PRPs on kn bits, and not on chosen ciphertext security. While a super-PRP on kn bits is desirable, it does not necessarily imply security when the adversary is allowed to submit ciphertexts of varying lengths. We note that RPC is not intended to be a super-PRP; instead, it is meant to give chosen ciphertext security via unforgeability.

One mode of encryption was suggested by Naor and Reingold [20]. They provide a construction which extends a block cipher on n-bits to a super-PRP on $2in$ bits, for any $i \geq 1$. However, it is unclear how to extend their construction to handle variable input lengths. Furthermore, the construction requires shared keys for both the underlying block cipher and two additional hash functions. Finally, since the construction requires two applications of a hash function to strings as long as the plaintext message, the construction is not inherently parallelizable.

A different mode of encryption was suggested by Bleichenbacher and Desai [8]. Their construction gives a super-PRP on messages of arbitrary length, and requires the parties to share keys to three underlying block ciphers. However, it is not clear (and we were unable to prove) that their mode is secure under chosen ciphertext attacks when the adversary is allowed to submit ciphertexts of varying lengths. Also, the scheme is relatively inefficient, as it requires three applications of the underlying block cipher for every block of the plaintext message, and the required computation is not parallelizable.

RPC mode is simple and leads to a straightforward security analysis. The drawback of the mode is the ciphertext expansion and resulting slowdown: for practical security one would want $r \geq 32$ which gives impractical expansion when using a 64-bit block cipher. When using block ciphers with 128-bit or larger block-sizes (e.g., AES), this is less of a concern (only 33% expansion). Advantages of RPC include the fact that the parties need share only one key and that the scheme is completely parallelizable. In contrast, authentication using a MAC requires an additional key and another cryptographic computation per block (which in many cases requires highly sequential computation). Minimizing shared key lengths is important in any integration of public-key and private-key encryption (such as e-mail encryption software). We further note that the ciphertext expansion in the current state of increased communication and storage bandwidth does not seem like a real limitation.

As an open research direction, we note that the mode presented here has the ciphertext a (constant) multiplicative factor longer than the plaintext. Can this be improved to obtain a provably secure, single-key mode in which the ciphertext is longer than the plaintext by only an additive constant?

References

1. ANSI X3.106, "American National Standard for Information Systems—Data Encryption Algorithm—Modes of Operation," American National Standards Institute, 1983.
2. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation. FOCS 1997.
3. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. CRYPTO 1998.
4. M. Bellare, O. Goldreich, and S. Goldwasser. Incremental Cryptography and Application to Virus Protection. STOC 1995.
5. M. Bellare and P. Rogaway. On the Construction of Variable-Input-Length Ciphers. FSE 1999.
6. E. Biham. Cryptanalysis of Multiple Modes of Operation. J. of Cryptology 1998.
7. E. Biham and L.K. Knudsen. Cryptanalysis of the ANSI X9.52 CBCM Mode. EUROCRYPT 1998.
8. D. Bleichenbacher and A. Desai. A Construction of a Super-Pseudorandom Cipher. Manuscript, February 1999.
9. D. Dolev, C. Dwork, and M. Naor. Non-malleable Cryptography. SIAM J. Computing, to appear; a preliminary version appears in STOC 1991.
10. S. Goldwasser and S. Micali. Probabilistic Encryption. JCSS, 28: 270-299, 1984.
11. ISO 8372, "Information Processing—Modes of Operation for a 64-bit Block Cipher Algorithm," International Organization for Standardization, Geneva, Switzerland, 1987.
12. M. Jakobsson, J.P. Stern, and M. Yung. Scramble All, Encrypt Small. FSE 1999.
13. C.J.A. Jansen and D.E. Boekee. Modes of Blockcipher Algorithms and Their Protection Against Active Eavesdropping. EUROCRYPT 1987.
14. C. Kaufman, R. Perlman, and M. Speciner. "Network Security: Private Communication in a Public World," Prentice Hall, New Jersey, 1995, pp. 89–92.
15. J. Katz and B. Schneier. A Chosen Ciphertext Attack Against Several E-mail Encryption Protocols. 9th USENIX Security Symposium, to appear.
16. J. Katz and M. Yung. Complete Characterization of Security Notions for Probabilistic Private-Key Encryption. STOC 2000.
17. J. Katz and M. Yung. Chosen-Ciphertext Secure Incremental Encryption. Manuscript, February 2000.
18. M. Luby. Chapter 14, "Pseudorandomness and Cryptographic Applications," Princeton University Press, 1996.
19. C.H. Meyer and S.M. Matyas. "Cryptography: A New Dimension in Computer Data Security," John Wiley & Sons, New York, 1982.
20. M. Naor and O. Reingold. On the Construction of Pseudorandom Permutations: Luby–Rackoff Revisited. STOC 1997; also: personal communication, December 1999.

21. M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks. STOC 1990.
22. National Bureau of Standards, NBS FIPS PUB 81, "DES Modes of Operation," U.S. Department of Commerce, 1980.
23. B. Preneel. Cryptographic Primitives for Information Authentication—State of the Art. State of the Art in Applied Cryptography, 1997.
24. B. Preneel, M. Nuttin, V. Rijmen, and J. Buelens. Cryptanalysis of the CFB Mode of the DES with a Reduced Number of Rounds. CRYPTO 1993.
25. C. Rackoff and D. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. CRYPTO 1991.

Efficient Methods for Generating MARS-Like S-Boxes

L. Burnett, G. Carter, E. Dawson, and W. Millan

Information Security Research Centre,
Queensland University of Technology,
GPO Box 2434, Brisbane 4001
Queensland, Australia
FAX: +61-7-3221 2384
Email: {burnett, dawson}@isrc.qut.edu.au

Abstract. One of the five AES finalists, MARS, makes use of a 9x32 s-box with very specific combinatorial, differential and linear correlation properties. The s-box used in the cipher was selected as the best from a large sample of pseudo randomly generated tables, in a process that took IBM about a week to compute. This paper provides a faster and more effective alternative generation method using heuristic techniques to produce 9x32 s-boxes with cryptographic properties that are clearly superior to those of the MARS s-box, and typically take less than two hours to produce on a single PC.

1 Introduction

The Data Encryption Standard [7] has, for the past 25 years, been the US standard for symmetric (shared-key) encryption. In recent years, however, its block and key length have proved to be incapable of providing the levels of security required for applications which utilise shared key encryption. The call for a new standard to replace the Data Encryption Standard for shared-key encryption has been a controversial issue within the cryptographic community for the past two years. The new standard to be known as the Advanced Encryption Standard (AES) [8] has currently been narrowed down to five candidates out of the fifteen initial submissions to the call for AES algorithms in 1997.

The security of a block cipher rests almost entirely on the strength of the components of which it is comprised. These components must not only be secure individually, but must also achieve a much higher level of security when organised together as a cipher system. Substitution boxes (s-boxes) are one of the most important components of a block cipher. They contribute a variety of strengthening properties to the cipher as a whole. Their basic mechanism of allowing bits coming in to an s-box to be replaced with bits going out of an s-box makes them an obvious (and often the only) means of providing nonlinearity to a cipher.

One of the five final candidates for the AES is the MARS cipher [3]. IBM, the designers of the MARS symmetric block cipher, have generated a 9x32 s-box which is used in various stages of its computations, both as a 9x32 s-box and as two 8x32 s-boxes. A large effort was undertaken on the part of the designers to generate an s-box which satisfied the very specific properties outlined in the MARS documentation. This required an approximate program execution time of "about a week". However, as will be shown in Section 3, the MARS s-box does not satisfy all the required properties.

In this paper we present an alternative approach to the generation of MARS-like s-boxes using a heuristic technique known as hill climbing. We discuss the cryptographic properties achieved by hill climbing, and in particular give a comparison between these and the property requirements of the MARS s-box. We show that in order for our hill climbing application to satisfy the requirements of the MARS s-box, the program execution time for generation of an s-box was *at most* 3.3 hours. The average generation time for a 9x32 MARS-like s-box using our approach was approximately 2 hours. Apart from speed, hill climbing provides individual output functions that have cryptographic properties superior to those of the MARS s-box output functions.

The remainder of this paper is set out as follows: In Section 2 we outline some important fundamentals in s-box theory. In Section 3 we comment on the cryptographic property requirements of the MARS s-box. Section 4 discusses the techniques used by the designers of MARS to generate the 9x32 s-box used in their computations. We also describe our alternative technique for generating MARS-like s-boxes which satisfy the same requirements imposed by the MARS designers. In Section 5 we discuss some possible variations to our generation technique and relationships between requirements for the MARS s-box and our s-box. Some concluding points are put forward in Section 6, together with some directions for future research in this area.

2 S-Box Fundamentals

An MxN substitution box (s-box) is a mapping from M input bits to N output bits. There are 2^M inputs and 2^N possible outputs for an MxN s-box. A 9x32 s-box, such as the MARS s-box, has 9 input bits and thus $2^9 = 512$ possible inputs. Each input maps to a 32-bit output word.

S-boxes can also be considered as an ordered set of single output boolean functions. The truth table of a boolean function $f(x)$ is a vector containing 2^M elements, each element $\in$ {0,1}. The polarity truth table of a boolean function, denoted $\hat{f}(x)$, is a simple translation from the truth table where every element 0 in the truth table is replaced by the element 1 in the polarity truth table and every element 1 in the truth table is replaced by the element -1 in the polarity truth table. The relationship can be defined as $\hat{f}(x) = 1 - 2f(x)$.

The hamming distance between two boolean functions $f(x)$ and $g(x)$ is the number of truth table positions in which they are different. The Walsh Hadamard Transform (WHT) of a boolean function, denoted $F(\omega)$, is defined as

$$F(\omega) = \sum_x f(x)L_\omega(x) \text{ where } L_\omega(x) \text{ is the linear function selected by } \omega$$

and gives a measure of the correlation between a boolean function and the set of all linear functions and their complements. Note that a linear function, $L(x) = \sum a_i x_i$ with $a_i \in \{0, 1\}$.

Boolean functions (and therefore s-boxes) are required to exhibit cryptographic properties in order for them to effectively resist certain cryptanalytic attacks. We briefly describe below some of these properties.

A boolean function of M input variables which contains 2^{M-1} ones in its truth table is said to be **balanced**. This property ensures that there is no bias in the truth table. The advantage of using balanced boolean functions is that they cannot be approximated by a constant function. Thus balance is a desirable property to achieve in boolean functions.

The **nonlinearity** of a boolean function $f(x)$ of M variables is given by

$$N_f = \frac{1}{2} \text{ x } (2^M - WH_{max})$$

where WH_{max} represents the maximum absolute value of the Walsh Hadamard Transform. The nonlinearity of a boolean function is the minimum Hamming distance to the set of all affine (linear) boolean functions. By this definition a boolean function with high nonlinearity cannot be well approximated by a linear function, thus making the function more resistant to linear cryptanalysis. For this reason, nonlinearity is considered to be one of the most important cryptographic properties of boolean functions.

The autocorrelation function, denoted $\hat{r}_f(s)$, of $\hat{f}(x)$, the polarity truth table of $f(x)$, can be expressed as

$$\hat{r}_f(s) = \sum_x \hat{f}(x)\hat{f}(x \oplus s).$$

This cryptographic property provides a measure of the imbalance of all first order derivatives of the boolean function $f(x)$. A boolean function with low autocorrelation makes it more resistant to differential cryptanalysis in that the lower the autocorrelation value, the more difficult it is to approximate the function's first order derivatives.

An M-variable function $f(x)$ is said to be k^{th} order correlation immune, denoted CI(k), if it is statistically independent of the subset $x_{i_1}, x_{i_2}, ..., x_{i_k}$ of input variables where $1 \leq k \leq M$. In terms of security, the output of a correlation immune boolean function, reveals no information about small subsets of input values.

As boolean functions are the building blocks of s-boxes, it is typical to require the same cryptographic properties to be present in s-boxes to improve their strength and ability to resist existing cryptanalytic attacks as well as possible future attacks.

3 MARS Property Requirements

3.1 MARS Differential Requirements

The designers of the MARS cipher, in designing their 9x32 s-box, placed particular emphasis on ensuring that their s-box satisfied a number of property requirements. In this subsection, we discuss these requirements with respect to the combinatorial/differential properties of the s-box and also point out where the s-box does not satisfy one of the stated properties.

Note that they have referred to their 9x32 s-box as S[i], $0 \leq i \leq 511$. This s-box may be divided into two 8x32 s-boxes, S0[j] and S1[j], where $0 \leq j \leq 255$. For ease, we will adopt this notation here also.

Differential Requirements from [3]

1. S does not contain either the word 0x00000000 (all zeros word) or the word 0xffffffff (all ones word).

2. Every pair of distinct entries in each of the two 8x32 s-boxes, S0 and S1, differs in at least three out of four bytes. Equivalently, a pair of words from the same 8x32 s-box may have no more than one byte the same, in the same position.

3. The 9x32 s-box, S, does not contain two entries S[i] and S[j] ($i \neq j$) such that:

i) $S[i] = S[j]$;
ii) $S[i] = \neg S[j]$; or
iii) $S[i] = -S[j]$.

In other words, there are no two entries in S which are (i) identical; (ii) are complements of each other; and (iii) sum modulo 2^{32} to give zero.

4. (i) The xor difference of each distinct pair of entries in S is unique and (ii) the subtraction difference of each distinct pair of entries in S is unique.

5. Each distinct pair of entries in S differs by at least four bits.

An examination of the way the s-box entries of MARS are incorporated into the cipher reveals why requirements 1 - 5 are important. An input value selects an s-box entry and this entry is either xored with, added modulo 2^{32} to, or subtracted modulo 2^{32} from an intermediate value. By excluding the all zero subblock, any xor operation involving an s-box entry and an intermediate value changes the intermediate value. In addition, exclusion of the all one subblock ensures that not all bits of an intermediate value are altered. Requirements 2 and 5 ensure that any two distinct entries in S are somewhat different at both the byte and bit level. Requirement 3 ensures that the effect of one s-box entry cannot be cancelled by another entry from the s-box. Requirement 4 ensures that each possible output "difference" of the s-box is equiprobable, i.e. the Difference Distribution Table is flat.

3.2 MARS Linear Requirements

In this subsection, we discuss the linear requirements imposed by the designers of MARS for their 9x32 s-box.

We note that the linear correlation properties of any MxN s-box can be considered as a (2^M)x$(2^N$-1) matrix where the columns are the Walsh-Hadamard transform vectors of the boolean functions formed by xoring all non-empty subsets of the output functions. Thus the linear requirements can be restated as bounds on the values taken in this linear correlation matrix [2]. Since M is large, even calculating this matrix is very expensive, however we may calculate any individual column we like. The MARS linear requirements are all bounds on particular column subsets of this matrix, which can be calculated easily. It should be noted that the vast majority of the s-box linear correlation columns are not considered in any way by the MARS linear correlation requirements.

In addition, there are strict limitations on the values that can be taken for the correlations between boolean functions, and hence also for the values of bias that can occur. The bias values for an M-input boolean function can only take rational values that are a multiple of 2^{-M}. Thus the choice of bias values $\frac{1}{30}$ and $\frac{1}{22}$ in the property requirements needs explanation.

<u>Linear Requirements</u> from [3]

1. Parity Bias:

The parity bias of S given by $|Pr_x[\text{parity(S[x])} = 0]$ - $\frac{1}{2}|$ is to be at most $\frac{1}{32} = 0.03125$.

This requirement is a bound on the magnitude of the imbalance of the boolean function formed by xoring all output functions. This property thus affects only one column of the linear correlation matrix, that column being the xor sum of all 32 output boolean functions.

2. Single-bit Bias:

The single-bit bias of S given by $|Pr_x[\text{S}[x]_i = 0]$ - $\frac{1}{2}|$ $\forall$ i is to be at most $\frac{1}{30} \approx 0.03333$.

This requirement places a bound on the magnitude of the imbalance for all of the individual output functions. Thus 32 columns of the linear correlation matrix are affected.

3. Two Consecutive Bits Bias:

The two consecutive bits bias of S given by $|Pr_x[\mathrm{S}[x]_i \oplus \mathrm{S}[x]_{i+1} = 0]$ - $\frac{1}{2}|$ $\forall$ i is to be at most $\frac{1}{30} \approx 0.03333$.

This requirement places a bound on the magnitude of the imbalance of boolean functions formed by the xor of two adjacent outputs. There are 31 of these pairs, hence 31 matrix columns are affected.

4. Single-bit Correlation

The single-bit correlation of S given by $|Pr_x[\mathrm{S}[x]_i = x_j]$ - $\frac{1}{2}|$ $\forall$ i,j is to be minimised. The single-bit correlation bias for the MARS s-box is less than $\frac{1}{22} \approx 0.04545$.

This requirement seeks to minimise the correlation that all output functions have with the individual input bits. This requirement affects 32 x 9 = 288 matrix columns.

In all, only a maximum of 352 matrix columns out of 2^{32} - 1 are considered by the MARS linear requirements. With these same requirements, we are able to show that our heuristic processes are able to generate better properties, much quicker.

3.3 Satisfaction of MARS Properties

We shall now discuss the extent to which the MARS s-box was able to achieve the above properties.

MARS S-Box, S comprised of S0 and S1

S satisfies differential conditions 1, 3 and 5. S0 and S1 both satisfy differential condition 2. S does not satisfy differential condition 4. In [3], the authors state that S has 130816 distinct xor-differences and 2 x 130816 distinct subtraction-differences. This is not the case. S has 130813 distinct xor-differences and 2 x 130808 distinct subtraction-differences as evidenced below. In each equation, the xor/subtraction difference of the indexed words on the left is equal to the xor/subtraction difference of the indexed words on the right.

S[27] $\oplus$ S[292] = S[101] $\oplus$ S[360]
S[27] $\oplus$ S[101] = S[292] $\oplus$ S[360]
S[27] $\oplus$ S[360] = S[101] $\oplus$ S[292]
S[13] - S[138] = S[364] - S[297]
S[13] - S[364] = S[138] - S[297]
S[19] - S[168] = S[509] - S[335]
S[19] - S[509] = S[168] - S[335]
S[49] - S[142] = S[97] - S[392]
S[49] - S[97] = S[142] - S[392]

S[333] - S[131] = S[211] - S[348]
S[333] - S[211] = S[131] - S[348]

The parity bias of S is 2^{-7}, as stated in the MARS paper, which is less than the threshold value of $\frac{1}{32}$. The single-bit bias of S is at most ≈ 0.033203 which is slightly less than the limit of $\frac{1}{30} \approx 0.033333$. The two consecutive bits bias of S is at most ≈ 0.032290 which is less than the bound of ≈ 0.033333. The maximum single-bit correlation bias of S is about $0.044922 < 0.0454545$, as stated in the MARS paper. Thus all linear conditions imposed by the designers of the MARS s-box are satisfied by S.

4 S-Box Generation Techniques

4.1 Summary of MARS S-Box Generation Techniques

As mentioned earlier, the MARS s-box is a 9x32 s-box containing 512 32-bit entries. The approach taken by the designers of the s-box was to generate the 9x32 s-box by using the well known SHA-1 (Secure Hash Algorithm-1) [9]. SHA-1 produces a 160-bit digest comprised of the concatenation of five 32-bit words. The input used for SHA-1 is the value 5i|c1|c2|c3 where i = 0..102, cj (j $\in$ 1,2) are the fixed constants

c1 = 0xb7e15162
c2 = 0x243f6a88

and c3 is allowed to vary until the first eight property requirements are satisfied. The value for c3 which minimises requirement nine is then the one chosen. Therefore, each entry of the 9x32 s-box, S is computed as follows:

S[5i+k] = SHA-1(5i|c1|c2|c3)$_k$

denoting the kth word of the output of SHA-1 (k = 0..4, i = 0..102).

The designers started the computational process with c3 = 0, increasing its value until the resulting s-box was found. Each value of c3 resulted in a 9x32 s-box which was divided into two 8x32 s-boxes. For each value of c3, the xor sum of distinct pairs in S0 and S1 was checked to see if it contained more than one zero byte. If this was the case, then S[i] was replaced by 3 $\cdot$ S[i] for one of the words S[i] in the pair. The new s-box was again tested for the 5 differential requirements and first 3 linear requirements. If this test was passed then the single-bit correlation was calculated. The final fixed constant value of c3 was 0x02917d59. This value was found to best minimize the single-bit correlation.

As stated in [3], the program for generating S ran for about a week, with the value of c3 increasing to 0x02917459 = $43\ 086\ 937_{10} < 2^{26}$. The MARS s-box can be found in [3].

4.2 Summary of our Techniques for Generating MARS-like S-Boxes

Our approach to generating MARS-like s-boxes is a flexible one which allows for much variation in parameters and heuristic methods used. The particular technique we chose was a heuristic method known as hill climbing [6].

4.2.1 Hill Climbing

Making small changes to the truth table of a boolean function produces one of three effects on the WHT of the function - the WHT values can decrease, remain unchanged or increase. In terms of properties such as nonlinearity, this means that the nonlinearity measure of the new boolean function resulting from the change can either become smaller, remain the same or become larger. Hill climbing takes advantage of this effect to optimise cryptographic properties of boolean functions (and thus s-boxes) by retaining a change which has brought about an improvement in a property value, such as nonlinearity. Such an improvement is incremental and consequently explains the analogy with climbing a hill.

Essentially, hill climbing involves the following steps:

1. Measure the property of concern for the original function.
2. Select a pair of elements to complement ensuring that the pair chosen consists of a zero and a one. (This ensures balance is maintained).
3. Measure the property of concern for the new function.
4. If the measure of the property in 3 is 'better' than the measure of the property of the original function, then accept this new function as the original function. If the measure of the property in 3 is worse, then retain the original function.
5. Repeat steps 2, 3 and 4 until a predetermined stopping criteria has been reached.

4.2.2 General Procedure

The technique we used to create our s-boxes began with the generation of random single-output balanced boolean functions. Each boolean function was hill climbed to reach a minimum nonlinearity value, a parameter allowed to vary for optimum results. The goal of this approach was to generate a set of 32 balanced boolean functions which not only each achieved the minimum nonlinearity value set by the user, but was also constrained by a maximum imbalance limit between pairs of boolean functions and was further constrained by a maximum deviation limit from CI(1).

A set of 32 boolean functions achieving these limits comprise a 8x32 s-box containing 256 words. Pairs of s-boxes of this size were combined to form a 9x32 s-box. It seemed less complicated to generate 9x32 s-boxes in this way due to the necessity of satisfying certain requirements placed on the 8x32 s-boxes individually. The s-box was then checked for the differential and linear requirements placed on the MARS s-box. In order to satisfy differential condition 2, it was necessary to modify a small number of bytes in each of the 8x32 s-boxes, typically in less than half a dozen entries, and re-checking that condition, particularly for previous pairs of entries. Similarly, the satisfaction of differential requirement 4 involved replacing a small number of entries in the 9x32 s-box. Subsequently, the new s-box was tested for all 9 conditions again. We ensured that the introduction of any replacement entries in the s-box did not destroy the balance property achieved by the initial functions.

4.3 Experimental Results

We stated in Section 3 the differential and linear requirements, together with threshold values set by the designers of MARS for their s-box. We shall now discuss the extent to which our s-box was able to achieve these properties.

Our S-Box, SB comprised of Sb1 and Sb2

Note that SB[i] where i = 0..511 is a 9x32 s-box, and Sb1[j], Sb2[j] where j = 0..255 are both 8x32 s-boxes. SB can be found in **APPENDIX A** and also at http://www.isrc.qut.edu.au/papers/2000/AppendixA.txt.

SB satisfies differential conditions 1, 3, 4 and 5. Sb1 and Sb2 both satisfy differential condition 2 of the MARS s-box requirements.

The parity bias of SB is 0.019531 which is less than the threshold value of $\frac{1}{32}$ = 0.03125. The single-bit bias of SB is zero. The absence of any single-bit bias in SB is due to the balance in each of the 32 boolean functions which comprise the s-box. The two consecutive bits bias of SB is at most ≈ 0.024462 which is less than the bound of ≈ 0.033333. The maximum single-bit correlation bias of SB is $0.03125 < 0.0454545$. Consequently, all linear conditions imposed by the designers of the MARS s-box are satisfied by our example 9x32 s-box, SB.

The achievement of these results depended largely on the three parameters used in our s-box generation program. For our experiments, we typically generated sets of 32 boolean functions with a minimum nonlinearity of 110, although we experimented with parameters above and below this value. A parameter value for minimum nonlinearity at around 108 produced 8x32 s-boxes in less than 10 minutes, while minimum nonlinearities of 112 for an 8x32 s-box took about 3 to 4 hours to generate. Our second parameter was a limit on the maximum imbalance between distinct pairs of boolean functions in the set. A typical parameter value for this limit used in our computations was 10. It was desirable to have a low imbalance between pairs of boolean functions which consequently had the effect of reducing the two consecutive bits bias condition imposed by the MARS s-box designers. We also placed a large degree of importance on our third parameter, the maximum deviation from CI(1). By minimizing this parameter value over all boolean functions, we were easily able to produce a single-bit correlation value below the given bound. Typically, we used parameter values such as 16 or 24 to be the maximum allowable deviation from CI(1) for the 32 boolean functions comprising the 8x32 s-boxes.

A combination of hill climbing and appropriate setting of the parameters discussed above allowed us to produce good 8x32 s-boxes, pairs of which gave us 9x32 s-boxes. Most of the s-boxes generated by our technique were very close to satisfying the MARS s-box requirements. In fact, for those s-boxes which we successfully generated, bias values, when exceeded, were so by only extremely small margins. The remaining s-boxes which we generated were easily able to satisfy the same conditions that the MARS s-box satisfies.

Based on a heuristic technique approach, we were able to generate a number of MARS-like s-boxes with little effort. In addition, the program execution time, depending on the parameters chosen, varied from approximately 16 minutes to around 3 hours and 20 minutes on a single Pentium II 300 MHz PC. This time frame is a huge improvement on the program running time for the MARS s-box of about a week.

5 Property Relationships and Technique Variation

5.1 Property Relationships

An s-box comprised of balanced boolean functions clearly possesses no single-bit bias since the number of ones and zeros in the truth table of balanced boolean functions is the same. Our s-box generation procedure began with the generation of a set of 32 balanced boolean functions. In order to satisfy differential requirements 2 and 4 it was necessary to replace a small number of bytes and entries respectively. However, at all times throughout our computations we retained balance in the boolean functions. None of the boolean functions comprising the MARS s-box are balanced. However, their deviation from balance is not large enough to violate the single-bit bias requirement.

Nonlinearity is a very important cryptographic property of single output boolean functions and s-boxes. Higher nonlinearity indicates a reduction in the magnitude of statistical correlations between sets of input bits and sets of output bits. The nonlinearity of an s-box is measured by the magnitude of the largest Walsh Hadamard Transform (WHT) value in the linear correlation matrix. The linear requirements in [3] are concerned solely with balance properties and no requirements on nonlinearity values are given. The nonlinearity of the individual boolean functions in Sb1 and Sb2 ranges from 108 to 112 inclusive, with an average nonlinearity of 110. The boolean functions comprising the MARS s-boxes, S0 and S1, have nonlinearity values ranging from 92 to 109, the most frequent nonlinearity value being 102.

Although in our s-box generation procedure we have not directly sought to optimise the autocorrelation property, the boolean functions comprising our s-boxes have, in general, displayed low autocorrelation values. A low autocorrelation distribution for an s-box serves to improve its differential properties, in particular, by flattening the Difference Distribution Table. The range of autocorrelation values for the individual boolean functions in our 8x32 s-boxes, Sb1 and Sb2, was between 48 and 88, averaging around 56. We note that the boolean functions comprising the MARS s-boxes, S0 and S1, displayed autocorrelation values of between 52 and 88, averaging 64.

One of the three important parameters set by our code was a limit for the maximum imbalance of the xor sum of distinct pairs of boolean functions. The purpose of this restriction on boolean function possibilities was to reduce the imbalance between pairs. A low level of imbalance between pairs of boolean functions includes the effect of reducing the two consecutive bits bias i.e. the bias between adjacent output boolean functions.

Our requirement for setting a maximum deviation from CI(1) for the individual boolean functions is identical to the single-bit correlation requirement placed on the MARS s-box. Minimising this measure reduces the magnitude of correlations which exist between individual input and individual output bits of the s-box.

5.2 Possible Variations on our Techniques

A great number of variations to our technique for generating MARS-like s-boxes may be adopted as alternative approaches to this task. An obvious generation method would be to apply another useful heuristic technique called the genetic algorithm to randomly generated boolean functions in order to "build" a cryptographically strong 9x32 s-box. Genetic algorithm applications have been very successful in improving cryptographic properties of boolean functions and s-boxes. Indeed, in [5] it was found that a combined genetic algorithm with hill climbing proved to be even more successful in generating boolean functions with good cryptographic properties such as nonlinearity and autocorrelation.

Additional parameters may be included in the code for the generation of a stronger s-box, for example, criteria for strict avalanche and propagation. Varying the parameters used in the generation process allows for a different strength emphasis in the resulting s-box, although the reader should note the existence of conflicting properties which affect each other in a negative way.

Further, it should be noted that only a small subset of the linear correlation matrix is utilised by the linear requirements imposed by the designers of the MARS s-box. However, as a consequence of our parameter choices a larger subset of the linear correlation matrix is utilised in the generation of our s-boxes, thus making greater use of the information contained in this matrix. We believe that an even stronger s-box can be generated if more information from this matrix is incorporated into s-box design. However, it should be noted that to generate the complete linear correlation matrix and analyse it in its entirety is not practical due to the computational effort required for this task.

6 Conclusions and Future Research

The designers of the MARS s-box have successfully generated a 9x32 s-box which satisfies all but one of the requirements placed on it relating to differential and linear properties. Their s-box failed to satisfy differential condition 4, despite claims that it did in fact do so. A long search running through values for c3 caused the program to take about a week to produce the final MARS s-box. In this paper we have presented an alternative approach to the generation of MARS-like s-boxes providing satisfaction of all of the requirements which were placed on the MARS s-box. Further, we have shown that by using a combination of random boolean functions, heuristic techniques and appropriate parameters we have gained additional properties such as higher nonlinearity and balance. This approach requires far less effort and compares very favourably to the MARS approach particularly in terms of computation time and ease of generating not only one but a number of MARS-like s-boxes.

Much work is needed to be done in this area in order to conduct an indepth investigation into the ways in which s-boxes with good cryptographic properties may be generated. The desirable properties are not only limited to those of the differential and linear type, even though their importance, stemming from targets of powerful cryptanalytic attacks, is by no means trivial. In the previous section we have endeavoured to outline a few of the variations on our techniques which could be investigated. Work directed towards other optimisation techniques for improving the cryptographic properties of s-boxes may be another worthwhile path for future research in this area. We believe that the strongest s-box will be one which achieves the correct balance between cryptographic properties. It is an open problem to find this balance.

References

1. E. BIHAM and A. SHAMIR Differential Cryptanalysis of DES-like Cryptosystems *Journal of Cryptology*, 4:3-72, 1991.
2. J. DAEMEN, R. GOVAETS and J. VANDEWALLE Correlation Matrices *Fast Software Encryption*, LNCS vol. 1008, pages 275-285, Springer-Verlag, 1994.
3. IBM Corporation MARS - a candidate cipher for AES *http://www.research.ibm.com/security/mars.html*.
4. M. MATSUI Linear Cryptanalysis Method of DES Cipher *Advances in Cryptology - Eurocrypt'93*, LNCS vol. 765, pages 386-397, Springer-Verlag, 1993.
5. W. MILLAN, A. CLARK and E. DAWSON An Effective Genetic Algorithm for Finding Highly Nonlinear Boolean Functions *International Conference on Information and Communications Security, ICICS '97 Lecture Notes in Computer Science* Vol. 1334, pages 149-158, Springer-Verlag, 1997.
6. W. MILLAN, A. CLARK and E. DAWSON Smart Hill Climbing Finds Better Boolean Functions *Workshop on Selected Areas of Cryptology, SAC '97, Proceedings*, pages 50-63, 1997.
7. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) Data Encryption Standard *U.S. Department of Commerce FIPS* Publication 46, January 1977.
8. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES) *Federal Register* Vol. 62 No. 177, pages 48051-48058.

9. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST) Secure Hash Standard *NIST FIPS PUB 180*, U.S. Department of Commerce, May, 1993.

APPENDIX A

An example S-box generated by our techniques using heuristic methods is SB[i] below where i = 0..511:

0x657ce571	0xb2c0a31b	0xeaaacac0	0xd4d49175	0x4794396c	0xada63322
0xb6476df8	0x5d8b1bdb	0x3216bd0c	0x87810f0e	0x8928aab6	0x309926d6
0x86ed7cda	0x7ce28025	0xab91f5e5	0xa0559c17	0x03b7fcc1	0xc635a7c2
0xb12e7967	0xf3c464ce	0x1c8815d1	0x12fa97fb	0x6937c3b8	0xd8f7406d
0x581a310f	0xf60add94	0x3e297a67	0x61ecf4be	0x4abcb39d	0x3fbf5af2
0xb01c48c9	0x01193559	0xae0d8259	0xd1229472	0x2a1c3b84	0xd3b54059
0xd1557eb8	0x6d1f101c	0xee7fd7ab	0x3ac220a1	0x03e23430	0xd6746be1
0x5e026256	0x57f98f80	0x8b09cf06	0xe503ea7f	0xa3268b2a	0x192bb9e3
0xc28a5f35	0x54c8ceef	0x0ee5da73	0x4d3154a7	0x8eda0ab3	0xe5e18c07
0x9e923d96	0xf94dc633	0xb02e60bc	0x1b6acf89	0xb8c718a2	0xad77b720
0x2444c1d0	0x9d64bd69	0x6c7ea648	0xc6b3f1be	0x85fbf907	0xb62ab1a0
0x105349ff	0x0c0d7808	0xb9af64fd	0x81f3c534	0x1a450da2	0xf5d20e38
0xd8ea00da	0xd149c90a	0x2b69526e	0x9d7d7598	0xefe96d87	0x7f55539a
0x819c2b62	0x7f85c4ff	0x6c8d1bb8	0xcf7b529d	0x664294e4	0x7eb2d2cc
0x6fd7ade7	0x4ee6b926	0xb858f38d	0xc2c47b42	0xbeb2006e	0x75003971
0x1eb2eb50	0x02eff63d	0x05dbe8ce	0x4d0ccac2	0x502fc81f	0x25724c59
0x9852165f	0xa9bd3bb2	0x40308156	0x319ebb09	0x3bb1370f	0x18718f78
0x751ed38a	0xe74acd36	0x59745744	0xda8f3b85	0xf4771cfe	0x6510184d
0xc36d332b	0xbfb8d681	0xe95e9ec7	0xc0332dec	0xcb24e5f4	0x6a746cbb
0xe9a5b509	0x0fbc5c93	0x8b138d45	0x8f6a906e	0xde78fe6b	0x131faa01
0xe79f8558	0x64b15239	0x255e0943	0x7be2d50a	0x6d28a6bc	0xba53449d
0x8cc7e39a	0xd29d82f2	0xd940cab8	0xe0d39beb	0xb079da15	0xdc1d313e
0xcb032e98	0x9e3ff5f0	0x77da39db	0x06cc4b3c	0x6d7323c6	0xc880d552
0x63fd8825	0x98e0d78d	0x6861c1cb	0x710fd4e5	0x79b69e4e	0x00061be5
0x623125bb	0xa54b082c	0xcdc97ce2	0x99f71a6f	0xd1443f73	0xb406ff77
0x04a2f4b4	0x67698cd3	0xaa3d5731	0x59c12151	0x5a9f8068	0xe29e555c
0xefacd992	0x418f3f8f	0xb3233fd9	0xe8c97421	0xe673f889	0x2fd7f4d8
0x5e838793	0x654e53b7	0x20fad86e	0x0729f2ce	0xf788004f	0xbcce24e0
0x1f27ab52	0x49ff2416	0xe6afc9b1	0x09995df5	0x834c7268	0x17daa0cf
0xacc21c23	0xb4f41552	0xf018c993	0xe247cf88	0x11caef8d	0xcab5a62f
0x41bf4a29	0x68147ece	0x16396c17	0x707d2204	0x74b40fac	0xde046da6
0xf2e39b32	0xafa3025d	0x18d2f854	0x1cb5d9ec	0x9fdb4066	0xd755650c
0xe178476e	0x81b6dadb	0x871587fe	0x0e4bfb09	0x7aad28c5	0xf32a077b
0xd3ee8184	0x7db97e78	0x77e03897	0x02d05ec4	0xe4daa729	0x94cedc15
0xc6a41eab	0x1499c20a	0x3f20e0d8	0xf22df536	0x2b2e1c53	0x104dac0e
0xc23faec3	0xacca64e3	0xaaf70012	0x3a498f24	0x21353c82	0x19a00a08
0x8d1016ed	0xa61b6b33	0x3743e626	0x5050a261	0x5dcb8660	0x7338f3a9
0x4e070f37	0xe9b2e637	0x779110a2	0x12792697	0x14457cbb	0x3884e0af
0x3e5a7cb8	0xf0b1a844	0xa4a05227	0x62637655	0x07e4e409	0xad8d8f5c
0x1de7a9fd	0xce4c62d9	0x3f08c7d5	0x5b56524f	0x98a46531	0x228b9fde
0xdc7e2e02	0x0ece49d2	0x853351ed	0xc1687310	0x3b92f374	0xf2cfa8dc

```
0x8eb66314  0x27ccb3f6  0xf7c9237e  0x4850359c  0x1954b0e1  0x1bb20c31
0x9525bed1  0x046e076b  0xdc54db4c  0x3bfca3b5
0x558773ab  0x28b2ad90  0xf6973f75  0xaca086bd  0xd7cdc737  0x56815e7d
0x1f0dbc30  0x78ace509  0xa6a0a02c  0x7fd04834  0xc9c4a588  0xee8681ee
0x3b7bb9c7  0x23ed1bd7  0xe26ef6ce  0x8e1f20db  0x1becd6e8  0x972421a6
0x37015bdf  0xd2fa8b20  0x98e1fa8c  0xf33a1d5e  0x2e40af46  0xad43454d
0xa12419d4  0x764b7009  0xf117276e  0x590e0446  0x9c9d0f79  0x464b03be
0x36ee34c1  0xcda77e71  0x318500d2  0x1e5e1033  0x902f9947  0xa98bc045
0xbee34bc2  0xee558b95  0xa033121d  0x299e0222  0xab2b7680  0xda013864
0x8456e39c  0x88f6d2c2  0x20387d0b  0xd59e867b  0xbdc85129  0x29f8a23d
0x05238897  0xa671d73a  0xe5a2749a  0x8498b1b9  0x5284c9a0  0x05a35fb5
0xfd641dc0  0x01e04f29  0x607c222e  0xe37daa81  0x3b504c33  0xd4283597
0xe4bf5c64  0x171271f7  0xb72de4be  0x85d072d7  0x95648343  0xc3fed911
0xe19d9084  0xfd46ddc1  0x5a28301f  0xc5106adc  0x7fdcafaa  0x97864b57
0x26441e1b  0x08fc5943  0x6bb68bf7  0xe0d10d7f  0x2ee6b878  0x62dc5145
0xbdbd3c1e  0x4fe9d606  0x44404830  0xfb8a3222  0xd3d97c07  0x6afd4cad
0x18772ce4  0x6b8a39a0  0x1853f57c  0xcf3894b1  0xae377739  0x1973945c
0x2c3fe2eb  0xd8b1f202  0x65b9cc80  0x11741355  0x8390161c  0x5fc3bf0a
0x3b98ec59  0xb2170ae3  0x04ff1c07  0x483272cc  0xd8e99670  0xec3e63c2
0xd5d8b58b  0x3ac2ec07  0x076be86e  0xa304eec4  0xea534d84  0x7a7a57e6
0xc106953c  0x5bb2ef63  0x06d3d71c  0x2a9fcad7  0x7ac837c9  0x620eeb1a
0xf4692142  0x1a906cab  0x71b25a75  0xd54dc1fe  0xe3791c6b  0x1779abcd
0x49c276fa  0x7735cfef  0xdc8fcaeb  0xb7e1bdc1  0x1e0a0ed4  0x674f3390
0xa506bc60  0x5409c03b  0xc08acd89  0xa01e5ee0  0x3d6b4a50  0x7d9d2477
0x32fabebe  0xba61f9d0  0x32d7f5c6  0x4d6d06f0  0x879ff997  0xa974cf79
0xd9727823  0x46918f88  0x0a19c4ff  0xfeb10189  0x09b4a7da  0x74f24b8c
0x1abd6bd9  0x2fee52ad  0x937b044c  0xfd8e3ccf  0x782d8e2b  0xc9dcd807
0x1ace6736  0xce30c05c  0xc0ab6563  0x98dd5602  0xb31a52a8  0xc3732b2e
0x88633833  0x4f0c5b26  0x30634892  0x6db73b5d  0x4099469f  0x79a5f0b6
0x9e7ecd2a  0x4177a990  0x71929643  0xfc9879e4  0x6f7c6158  0x31bc8ddc
0xda5a8574  0xc64df673  0xfeb43a6d  0x87caf074  0x1c8376ca  0x1a28b32f
0xd6d4e0b3  0x45682e84  0x21d2ce9d  0x72a7fc5e  0x58e87ae0  0x5404a615
0x86244304  0xeaeac14f  0x6b40d113  0xed15e3ab  0xc29f99f9  0x2c75e7ee
0xf364db5f  0x14e1058b  0xee592f43  0xa5e39cb3  0x884bfa85  0x120c3c1f
0x756b17b5  0x375c388b  0x8c1e7fea  0x0113d4e1  0x4b97581b  0xb243ef45
0xd7668abf  0xa32565b2  0x25bd9341  0xc1817258  0xde4136a8  0x5b5994d2
0xd55aaf72  0x2a1e200d  0xb320a251  0xb0c9d7bc  0xb0d9a994  0xd4fae0b9
0xc808f572  0x2cd72112  0xfd3dcd52  0x308db7b4  0x67989372  0xe847aca3
0x2f064d6f  0x34d14d7c  0x74b5a608  0xb7abf1d3  0x6ae58795  0xcc4fb967
0x44fa099c  0xbb56d026  0x8c3d93f7  0xde33075c  0x0221622a  0x592255ea
0xa1cefd2e  0xababb33d  0xc64f1a49  0x7e301a30  0x398361fb  0xb44f067c
0x0bdaa3a6  0x1d09ecb9  0x146ea2e5  0xd7b7508a  0x48656859  0x67ec3ff0
0x02ef76a8  0x8d97a9ee  0x0edc95e5  0x88be0eb7  0x11e59acf  0x4674ffed
0xb3966531  0x8d7798b8  0x6d021a5e  0xdb86e411  0x78618b37  0xafe9287c
0x9bd7274c  0xdd240d88  0x4412d92a  0x932082c9
```

Author Index