Lecture Notes in Computer Science 2355

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

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

Mitsuru Matsui (Ed.)

Fast Software Encryption

8th International Workshop, FSE 2001
Yokohama, Japan, April 2-4, 2001
Revised Papers

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editor

Mitsuru Matsui
Mitsubishi Electric Corporation
5-1-1 Ofuna Kamakura Kanagawa, 247-8501, Japan
E-mail: matsui@iss.isl.melco.co.jp

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Fast software encryption : 8th international workshop ; proceedings / FSE 2001, Yokohama, Japan, April 2 - 4, 2001. Mitsuru Matsui (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2355)
ISBN 3-540-43869-6

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

ISSN 0302-9743
ISBN 3-540-43869-6 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by PTP Berlin, Stefan Sossna e. K.
Printed on acid-free paper SPIN 10870083 06/3142 5 4 3 2 1 0

Preface

Fast Software Encryption is an eight-year-old workshop on symmetric cryptography, including the design and cryptanalysis of block and stream ciphers, as well as hash functions. The first Fast Software Encryption Workshop was held in Cambridge in 1993, followed by Leuven in 1994, Cambridge in 1996, Haifa in 1997, Paris in 1998, Rome in 1999, and New York in 2000. This Fast Software Encryption Workshop, FSE 2001, was held from 2-4 April 2001 in Yokohama, Japan, in cooperation with the Institute of Industrial Science, of the University of Tokyo.

This year a total of 46 papers were submitted to FSE 2001. After a two-month review process, 27 papers were accepted for presentation at the workshop. In addition, we were fortunate to be able to organize a special talk by Bart Preneel on the NESSIE project, a European initiative to evaluate cryptographic algorithms. The committee of this workshop was:

General Chair
Hideki Imai (The University of Tokyo)

Program Committee

Ross Anderson (Cambridge Univ.)	Eli Biham (Technion)
Cunsheng Ding (Singapore)	Henri Gilbert (France Telecom)
Dieter Gollman (Microsoft)	Thomas Johansson (Lund Univ.)
Lars Knudsen (Bergen Univ.)	James Massey (Denmark)
Mitsuru Matsui (Mitsubishi Electric, Chair)	Kaisa Nyberg (Nokia)
Bart Preneel (Katholieke Univ. Leuven)	Bruce Schneier (Counterpane)

We would like to thank all submitting authors and the committee members for their hard work. We are also appreciative of the financial support provided by Mitsubishi Electric Corporation. Special thanks are due to Toshio Tokita, Junko Nakajima, Yasuyuki Sakai, Seiichi Amada, Toshio Hasegawa, Katsuyuki Takashima, and Toru Sorimachi for their efforts in making the local arrangements for this workshop.

We were very pleased and honored to host the first FSE workshop held in Asia. Finally, we are also happy to announce that the next FSE will be the first FSE workshop sponsored by International Association for Cryptologic Research (IACR).

May 2002 Mitsuru Matsui

Table of Contents

Cryptanalysis of Block Ciphers II

Cryptanalysis of Stream Ciphers II

Pseudo-Randomness

FSE 2001 Special Talk

Cryptanalysis of Block Ciphers III

Design and Evaluation

The Saturation Attack – A Bait for Twofish

Stefan Lucks*

Theoretische Informatik
University of Mannheim, 68131 Mannheim, Germany
lucks@th.informatik.uni-mannheim.de

Abstract. This paper introduces the notion of a "saturation attack". Consider a permutation p over w-bit words. If p is applied to all 2^w disjoint words, the set of outputs is exactly the same as the set of inputs. A saturation attack exploits this fact. The current paper applies saturation attacks on reduced-round variants of the Twofish block cipher with up to seven rounds with full whitening or eight rounds without whitening at the end (i.e., half of the cipher). The attacks take up to 2^{127} chosen plaintexts (half of the codebook) and are 2–4 times faster than exhaustive search. The attacks are based on key-independent distinguishers for up to six rounds of Twofish, making extensive use of saturation properties.

1 Introduction

Modern b-bit block ciphers often use permutations $p : \{0,1\}^w \to \{0,1\}^w$ with $w < b$ as building blocks. E.g., p may be an S-box, a round function, or a group operation where one of the operands is constant. The constant may be unknown to the cryptanalyst, e.g. as a part of the (round) key. We regard the input for p as a data channel. For the cryptanalyst, p may be known or unknown, and the cryptanalysts may be unable to determine the input for p. A *"saturation attack"* is based on the idea of choosing a set of $k * 2^w$ plaintexts such that each of the 2^w inputs for p occurs exactly k times. In this case, we say that the data channel into p is *"saturated"*. A saturation attack exploits the fact that if the input for p is saturated, then the output from p is saturated, too.

The name "saturation attack" is new, but such attacks have been studied before. E.g., the "Square attack" is a saturation attack, developed for the block cipher Square [4]. It works as well for other Square-like ciphers such as the AES candidate Crypton [11,12] and the finalist Rijndael [5], which has recently been chosen as the AES. All these ciphers are 128-bit block ciphers with 8-bit data channels. The attack starts with a set of 2^8 plaintexts with one saturated channel. The other 15 channels are constant. After two rounds, all 16 data channels are saturated. After three rounds, the saturation property is likely to have been lost, but the sum of all values in a data channel is zero. This allows to distinguish the three-round output from random. The best currently known attacks on Crypton [3] and Rijndael/AES [7] are extensions of the Square attack.

* Supported by German Science Foundation (DFG) grant KR 1521/3-2.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 1–15, 2002.

"Miss in the middle" attacks [1] are rudimentarily related to saturation attacks, exploiting the fact that given two inputs $x \neq y$ for a permutation p one gets two outputs $p(x)$ and $p(y)$ with $p(x) \neq p(y)$. Also related is the attack on "Ladder DES", based on choosing $c*2^{32}$ distinct inputs for a 64-bit data channel and checking if all the outputs are distinct [2].

When using "higher-order differentials" [9], one chooses a certain complete set of plaintexts and, after some rounds of the cipher, predicts a key-independent property with probability one. This resembles the current approach.

This paper shows that saturation attacks are a useful tool for ciphers which are definitely not Square-like. We concentrate on the AES finalist Twofish [15]. So far, the authors of Twofish published some preliminary cryptanalytic results [16,6] themselves, a key separation property has been identified for Twofish [13, 14,8], and some observations on the generation of the Twofish S-Boxes and on differential cryptanalysis have been made [10].

The motivation for this research has been twofold. First, even though Twofish has not been chosen as the AES, it is (and probably will continue to be) used in practice. E.g., recent versions of popular email encryption programs, namely PGP and GnuPG [17], implement Twofish. Second, the study of saturation attacks appears to be of independent interest in cryptanalysis.

1.1 Notation

We will use the notion of a "multiset" to describe a w-bit data channel. A multiset with $k * 2^w$ entries is *"saturated"* if every value in $\{0,1\}^w$ is found exactly k times in the multiset. If $k = 1$, a saturated multiset is the set $\{0,1\}^w$.

In the context of this paper, a data channel is always 32 bits wide, and we call a value in a data channel a *"word"*. We interchangeably view a word x as a 32-bit string $x = (x_{31}, \ldots, x_0) \in \{0,1\}^{32}$ and as an unsigned integer $x = \sum_i x_i * 2^i$. The addition of values in a data channel is thus addition mod 2^{32}. We write "$x \lll b$" for the rotation of the word x by b bits to the left, and "$x \ggg b$" for rotation to the right. E.g. $(x \lll b) \ggg b = x$ for all x and b, and $(x_{31}, x_{30}, \ldots, x_1, x_0) \lll 1 = (x_{30}, \ldots, x_1, x_0, x_{31})$. $\mathrm{LSB}(x) = x \bmod 2$ denotes the *"least significant bit (LSB)"* of x, and $\mathrm{LSB}^1(x) = \mathrm{LSB}(x \operatorname{div} 2)$ denotes the 2nd-least significant bit. Similarly, we define the *"most significant bit (MSB)"*: $\mathrm{MSB}(x) = \mathrm{LSB}(x \lll 1)$. If the multiset M denotes a data channel, the bits at the LSB-position of M are *"balanced"* if $\bigoplus_{m \in M} \mathrm{LSB}(m) = 0$. It turns out to be useful to also consider *"semi-saturated"* data channels. The multiset M is semi-saturated if one bit of M is constant and each of the 2^{31} remaining values for M appears exactly $2k$ times in M.

2 A Description of Twofish

In this section, we describe the structure of Twofish. We omit many details, concentrating on the properties of Twofish which are relevant for our attack.

2.1 The Main Operations of Twofish

Twofish is based on the following operations:

Whitening. Decompose a 128-bit text block into words $a_0, \ldots, a_3 \in \{0,1\}^{32}$. The Twofish whitening operation is the XOR of four key words $K_{j+\delta} \in \{0,1\}^{32}$ to the words a_j: $b_j := a_j \oplus K_{j+\delta}$ for $j \in \{0, \ldots, 3\}$, see Figure 1.

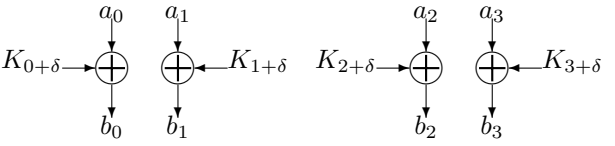


Fig. 1. The Twofish Whitening Operation.

Application of the round function. To compute the i-th round function F_i, use a pair $(a,b) \in (\{0,1\}^{32})^2$ as the input and and compute a pair $(a',b') = F_i(a,b) \in (\{0,1\}^{32})^2$. The round function F_i is defined by two round keys K_{2i+2} and K_{2i+3} and two functions $G_1, G_2 : \{0,1\}^{32} \to \{0,1\}^{32}$, see Fig. 2:

$$a' := G_1(a) + G_2(b) + K_{2i+2}, \quad \text{and} \quad b' := G_1(a) + 2G_2(b) + K_{2i+3},$$

The functions G_1 and G_2 are key-dependent, but do not depend on i. Given the round function's results a' and b', the remaining two words $c, d \in \{0,1\}^{32}$ come into play:

$$x := (a' \oplus c) \ggg 1, \quad \text{and} \quad y := b' \oplus (d \lll 1).$$

Except for the rotate operations, Twofish works like a Feistel cipher.

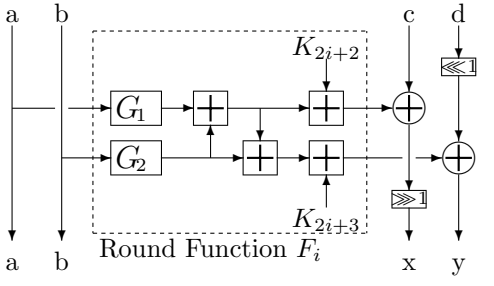


Fig. 2. The Application of the Twofish Round Function.

Our attack greatly depends on the functions G_1 and G_2 to be permutations over $\{0,1\}^{32}$. Actually, $G_2(x) = G_1(x \lll 8)$. Apart from that, the internal structure of G_1 and G_2 is not relevant for us.

The swap. Replace $(a,b,c,d) \in (\{0,1\}^{32})^4$ by (c,d,a,b). See Figure 3.

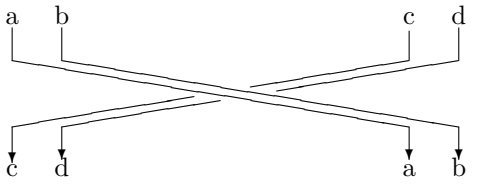


Fig. 3. The Twofish Swap.

2.2 The Basic Structure of Twofish

Twofish uses a 16-round Feistel structure with two additional one-bit rotates in each round, pre-whitening before the first round and post-whitening after the last round. Twofish works as follows:

1. Generate the key-dependent S-boxes, which define the functions G_1 and G_2.
2. Generate four subkey words $K_0, \ldots K_3 \in \{0,1\}^{32}$ for the pre-whitening, two subkey words K_{2i+2}, K_{2i+3} for each round and another four subkey words $K_{36}, \ldots, K_{39}$ for the post-whitening.
3. Given a plaintext block, do the pre-whitening.
4. For $i := 1$ to 15 do: (a) Apply the round function F_i.
   (b) Do the swap.
5. Apply the last round function F_{16} (no swap in the final round).
6. Do the post-whitening.

The first two of the above steps constitute the "key schedule" described below. Note that we can obviously generalise the Twofish structure to r rounds, where the loop in step 4 is iterated $r-1$ times.

2.3 The Twofish Key Schedule

A Twofish key consists of 128, 192, or 256 bit[1]: $2k$ words $M_0, \ldots, M_{2k-1} \in \{0,1\}^{32}$ with $k \in \{2,3,4\}$, organised as two vectors $M_e = (M_0, M_2, \ldots, M_{2k-2})$ and $M_o = (M_1, M_3, \ldots, M_{2k-1})$. A third vector $S = (S_0, S_1, \ldots, S_{k-1})$ is derived from M_e and M_o by using techniques from the theory of Reed-Solomon codes. Given any two of the three vectors M_e, M_o and S, the third one is easy to find.

With these three vectors, the "three halves of a Twofish key", we can do the first two steps of the structure described above:

1. The vector S determines the internal S-boxes and thus the functions G_1 and G_2. S is a k-word vector, while the key consists of $2k$ words or $64k$ bit.
2. The 40 subkey words $K_0, \ldots, K_{39}$ are defined by using functions h_e and h_o and by doing 20 *"subkey generation"* steps ($j \in \{0, \ldots, 19\}$):
   $A_j := h_e(j, M_e)$; $\quad K_{2j} := A_j + B_j$;
   $B_j := h_o(j, M_o)$; $\quad K_{2j+1} := (A_j + 2B_j) \lll 9$.

[1] These are the three generic key lengths of Twofish. Other keys of less than 256 bit are padded to the next generic length by appending zeros.

3 Distinguishers for Twofish

Given a well-chosen set of plaintexts, we describe how to distinguish reduced-round versions of Twofish from random permutations.

3.1 A Four-Round Distinguisher

Consider 2^{32} plaintexts $(\alpha_0, \alpha_1, A, \alpha_3)$, where α_0, α_1, and α_3 are three arbitrary 32-bit constants and A is the set of all 2^{32} words. The pre-whitening changes this set of texts to $(\beta_0, \beta_1, A, \beta_3)$ with new constants β_i.

Given this set of texts as the input for the first round, the input for the round function F_1 is constant: (β_0, β_1). By (γ_0, γ_1) we denote the output of F_1, which then generates the texts $(\beta_0, \beta_1, A, \gamma_3)$ with $\gamma_3 = (\beta_3 \lll 1) \oplus \gamma_1$. (Note that $A = \{a_i\} = \{(a_i \oplus \gamma_0) \ggg 1\}$.) The swap changes these texts to $(A, \gamma_3, \beta_0, \beta_1)$.

In the second round, the 2^{32} inputs for the round function are (A, γ_3). The round function generates the pairs (b_i, c_i) with $b_i = G_1(a_i) + G_2(\gamma_3) + K_6$ and $c_i = G_1(a_i) + 2G_2(\gamma_3) + K_7$ for $a_i \in A$. The sets $B = \{b_i\}$ and $C = \{c_i\}$ are saturated, just like A. Applying the round function here means XORing the constant β_0 to the values of B, followed by a rotation, and XORing $\beta_1 \lll 1$ to C. Neither operation changes the saturated sets B and C. We get 2^{32} texts (A, γ_3, B, C), where A, B, and C are saturated. By the swap, we get texts (B, C, A, γ_3).

The 2^{32} inputs for the third round function are of the form (B, C) with saturated B and C. Since both G_1 and G_2 are permutations, $G_1(b_i) \neq G_1(b_j)$ and $G_2(c_i) \neq G_2(c_j)$ for $b_i, b_j \in B$, $c_i, c_j \in C$, and $i \neq j$. Let $d_i = G_1(b_i) + G_2(c_i) + K_8$ and $e_i = G_1(b_i) + 2G_2(c_i) + K_9$. The 2^{32} outputs of the round function are of the form (D, E), with the multisets $D = \{d_i | 0 \leq i < 2^{32}\}$ and $E = \{e_i | 0 \leq i < 2^{32}\}$. Neither D nor E is likely to be saturated. However, we are still able to observe a weaker property: Since $\sum_{b_i \in B} b_i = \sum_{c_i \in C} c_i = \sum_{0 \leq i < 2^{32}} i \equiv 2^{31} \bmod 2^{32}$:

$$\sum_{0 \leq i < 2^{32}} d_i \equiv 2^{31} + 2^{31} + 2^{32} * K_8 \equiv 0 \pmod{2^{32}},$$

$$\sum_{0 \leq i < 2^{32}} e_i \equiv 2^{31} + 2 * 2^{31} + 2^{32} * K_9 \equiv 2^{31} \pmod{2^{32}},$$

thus $\sum d_i \equiv \sum e_i \equiv 0 \pmod 2$ – i.e., the LSBs of D and E are *balanced.*

Applying the round function means to evaluate 2^{32} pairs (f_i, g_i) with $f_i = f_i' \ggg 1$, $f_i' = a_i \oplus d_i$, and $g_i = (\gamma_3 \lll 1) \oplus e_i$. Define the multisets $F = \{f_i\}$, $F' = \{f_i'\}$, and $G = \{g_i\}$. We observe: The bits at the LSB-positions of both F' and G are balanced, and, due to the rotate, the bits at the MSB-position of F are balanced. Hence, the third round generates 2^{32} texts of the form (B, C, F, G), which are then swapped to (F, G, B, C).

The multisets (F, G) of inputs for the fourth round function F_4 are balanced. We write $(?, ?)$ for the outputs. Applying F_4 gives us 2^{32} texts $(F, G, ?, ?)$. After the swap, we get $(?, ?, F, G)$, where one bit in each F and G is balanced.

Figure 4 describes graphically, how the distinguisher works. Having chosen 2^{32} plaintexts, we can check the balancedness of the ciphertext bits at the two positions determined by the MSB of F and the LSB of G. Whatever the keys are, four rounds of Twofish always pass this test – even the post-whitening cannot destroy the balancedness. But a random permutation only passes this test with about a 25% probability.

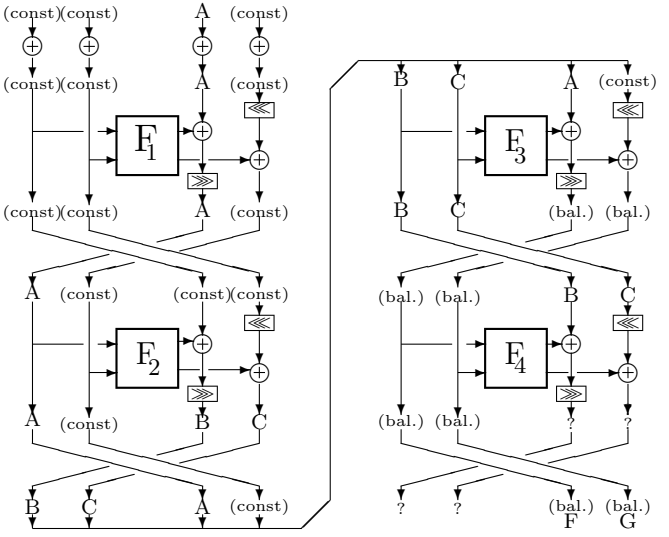


Fig. 4. The Four-Round Distinguisher from Section 3.1

3.2 Another Four-Round Distinguisher

Our second distinguisher works quite similarly to the first one. We start with 2^{32} plaintexts of the form $(\alpha_0, \alpha_1, \alpha_2, A)$ with arbitrary constants α_i. The pre-whitening changes the constants and we get texts $(\beta_0, \beta_1, \beta_2, A)$. After the first round, including the swap, these are $(\gamma_2, A, \beta_0, \beta_1)$.

In the second round, the inputs to the round function are of the form (γ_2, A), where $A = \{0,1\}^{32}$ is a saturated set and γ_2 is constant. The round function generates the pairs (b_i, c_i) with $b_i = G_1(\gamma_2) + G_2(a_i) + K_6$ and $c_i = G_1(\gamma_2) + 2G_2(a_i) + K_7$ for $a_i \in A$. Now the set $B = \{b_i\}$ is saturated like A, but the multiset $C^* = \{c_i\}$ isn't. Instead, it is semi-saturated with a constant $\mathrm{LSB}(c_i) = \gamma^* \in \{0,1\}$ for all $c_i \in C^*$: $\gamma^* = \mathrm{LSB}(G_1(\gamma_2)) \oplus \mathrm{LSB}(K_7)$. We apply the round function by adding some constants to the elements of B and C^*, and by then rotating the elements of B. The results are a saturated set B and a semi-saturated set C^*, as before. After the swap, we have texts of the form (B, C^*, γ_2, A).

In the third round the 2^{32} inputs for the round function are of the form (B, C^*). Consider the round function's outputs (d_i, e_i) with $d_i = G_1(b_i) +$

$G_2(c_i) + K_8$ and $e_i = G_1(b_i) + 2G_2(c_i) + K_9$. Since $B = \{b_i\}$ is saturated, so is $\{G_1(b_i)\}$, and especially

$$\sum_{0 \le i < 2^{32}} G_1(b_i) \equiv 0 \pmod 2.$$

Since C^* is semi-saturated, it has 2^{31} different values, each repeated exactly twice. The same holds for the 2^{32} values $G_2(c_i)$ (with $c_i \in C^*$), hence

$$\sum_{0 \le i < 2^{32}} G_2(c_i) \equiv 0 \pmod 2.$$

Thus, both multisets $D = \{d_i\}$ and $E = \{e_i\}$ are balanced:

$$\sum_{0 \le i < 2^{32}} d_i \;=\; \sum_i G_1(b_i) + \sum_i G_2(c_i) + 2^{32} * K_8 \equiv 0 \pmod 2$$

and

$$\sum_{0 \le i < 2^{32}} e_i \;=\; \sum_i G_1(b_i) + 2 * \sum_i G_2(c_i) + 2^{32} * K_9 \equiv 0 \pmod 2.$$

By applying the round function and swapping, we get 2^{32} texts of the form (F, G, B, C^*). The bits at the LSB-position of G are balanced, as are the bits at the MSB-position of F (due to the one-bit rotate). The fourth round makes this $(F, G, ?, ?)$, and if we do the swap we get texts of the form $(?, ?, F, G)$.

A random permutation passes the corresponding test only with a probability of about 0.25.

3.3 An Extension to Five Rounds

Next, we show how to extend the distinguisher from Section 3.2 to five rounds. Let α an arbitrary 32-bit constant and c^* an arbitrary 1-bit constant. We choose all 2^{95} plaintexts of the form (α, a_i, b_j, c_k), with $c_i \operatorname{div} 2^{31} = c^*$. We write (α, A, B, C^+) for these 2^{95} texts. Note that the multisets A and B are saturated and the multiset C^+ is semi-saturated. The pre-whitening changes the constant α to β, and the constant c^* to γ^*, but leaves A and B saturated and C^+ semi-saturated with a constant MSB. We still have 2^{95} distinct input texts (β, A, B, C^+) for the first round.

Let $(e_i, f_i) = F_1(\beta, a_i)$ with $a_i \in A$. We can write $e_i = \beta_e + G_2(a_i)$ and $f_i = \beta_f + 2G_2(a_i)$, for some constants β_e, β_f. Hence the outputs of F_1 consist of pairs (E, F^*) with saturated E and semi-saturated F^*. Set $\beta^* = f_i \bmod 2$ for the constant LSB of the values $f_i \in F^*$.

For every value $a_i \in A$ there are 2^{63} pairs (b_i, c_i) with a constant bit $\gamma^* = c_i \operatorname{div} 2^{31} = \mathrm{MSB}(c_i)$. We can fix any constants $\gamma_2, \gamma_3 \in \{0,1\}^{32}$ with $\gamma_3 \bmod 2 = \gamma^* \oplus \beta^*$ and find pairs (b_i, c_i) in (B, C^+) such that $(e_i \oplus b_i) \ggg 1 = \gamma_2$ and $f_i \oplus (c_i \lll 1) = \gamma_3$ holds for every a_i. (Note that the MSB of c_i is the LSB of $c_i \lll 1$.)

Now the 2^{95} input texts (β, A, B, C^+) can be separated into 2^{63} disjoint groups of 2^{32} texts, determined by the pair (γ_2, γ_3) of constants, such that after applying the first round functions all texts in the same group are of the form $(\beta, A, \gamma_2, \gamma_3)$. The swap changes these to $(\gamma_2, \gamma_3, \beta, A)$.

For each such group, applying the four-round distinguisher from Section 3.2 would result in a set of 2^{32} ciphertexts $(?, ?, F, G)$, where the ciphertext bits at the LSB-position of G and at the MSB-position of F are balanced. Now, we do not know which ciphertexts belong into which group, but if these bits for each group are balanced, then so are all 2^{95} such bits. Five rounds of Twofish always pass this test, while a random permutation passes it with about 25% probability.

The same technique can also be applied to the distinguisher from Section 3.1. Here, we need 2^{96} plaintexts of the form (α, A, B, C) with constant α. A random permutation passes the corresponding test with about 25% probability.

3.4 An Extension to Six Rounds

To attack six rounds, we choose 2^{127} plaintexts (a_i, b_i, c_i, d_i), (half of the codebook (!)), where $b_i \operatorname{div} 2^{31} = \mathrm{MSB}(b_i)$ is fixed to an arbitrary constant. Our plaintexts are of the form (A, B^+, C, D), where A, B, and D are saturated multisets, and B^+ is a semi-saturated one.

Our choice of plaintexts ensures that for each of the 2^{63} left-side pairs (a_i, b_i), all 2^{64} right-side pairs (c_i, d_i) exist. Neither the pre-whitening nor the application of the first round function change this property. By the swap we get 2^{127} texts (C, D, A, B^+) as the input for the second round. For each 32-bit constant α we get a group of 2^{95} texts (α, D, A, B^+). These are 2^{32} disjoint groups which are the kind of input we need for the 5-round distinguisher.

After six rounds of Twofish, we get 2^{127} ciphertexts $(?, ?, F, G)$ with balanced bits at two positions. A random permutation does satisfy this with about 25% probability.

3.5 Distinguishers: Summary

In Table 1 we summarise the distinguishers we have found. We describe which section the distinguisher was described in, the number r of Twofish rounds the attack works for, the chosen plaintexts required (how they look like and how many we need), and the probability for a random permutation to pass the test. All tests are one-sided, i.e. r rounds of Twofish pass the test with probability 1.

4 Finding the Key

In modern cryptanalysis, one often uses a distinguisher for some rounds of a product cipher to find the key: Guess some key bits for one or more additional rounds and exploit the distinguishing property to falsify wrong key guesses. This is what we do below, concentrating on using the six-round distinguisher.

Table 1. Distinguishers for Twofish.

Section	Rounds	Chosen Plaintexts		Probability
	r	Form	Number	
3.1	4	$(\alpha_0, \alpha_1, A, \alpha_3)$	2^{32}	25%
3.2	4	$(\alpha_0, \alpha_1, \alpha_2, A)$	2^{32}	25%
3.3	5	(α, A, B, C^+)	2^{95}	25%
3.3	5	(α, A, B, C)	2^{96}	25%
3.4	6	(A, B^+, C, D)	2^{127}	25%

4.1 The Basic Technique

Consider seven rounds of Twofish. Let 2^{127} plaintexts be chosen as required for the six-round distinguisher. After six rounds (including the swap), we have 2^{127} text quadruples (a_i, b_i, c_i, d_i) of 32-bit words. We have two distinguishing properties: the bits at the LSB-position of the words d_i are balanced, and the bits at the MSB-position of the c_i-words are balanced. We start with concentrating on the first distinguishing property.

The XOR over the 2^{127} LSBs $\delta_i = d_i \bmod 2$ of d_i is:

$$\delta^* = \bigoplus_{0 \le i < 2^{127}} \delta_i = 0.$$

After seven rounds, we have 2^{127} ciphertext quadruples (w_i, x_i, y_i, z_i), and we cannot count on balanced ciphertext bits. Note that the seventh round uses the keys K_{16} and K_{17}, and the post-whitening keys denoted by $K_{18}, \ldots, K_{21}$. Figure 5 visualises the last round, including the post-whitening.

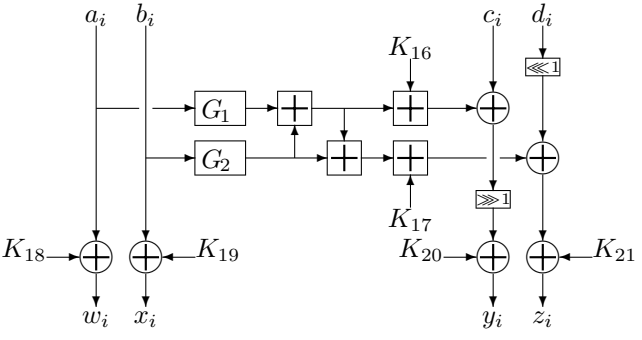


Fig. 5. The Seventh Round and the Post-Whitening.

Rewrite δ_i by $\delta_i = \delta_{w,i} \oplus (\delta^{\mathrm{c}}_{w,i} * \mathrm{LSB}(K_{17})) \oplus \delta_{x,i} \oplus \delta_{z,i} \oplus \mathrm{LSB}^1(K_{17})$ with

$$\begin{aligned}
\delta_{w,i} &= \mathrm{LSB}^1(G_1(w_i \oplus K_{18})) \\
\delta^{\mathrm{c}}_{w,i} &= \mathrm{LSB}(G_1(w_i \oplus K_{18})), \\
\delta_{x,i} &= \mathrm{LSB}(G_2(x_i \oplus K_{19})), \\
\delta_{z,i} &= \mathrm{LSB}^1(z_i) \oplus \mathrm{LSB}^1(K_{21}).
\end{aligned}$$

The necessity to consider the 2nd-least significant bits $\mathrm{LSB}^1(\ldots)$ is due to the last round's one-bit rotate: $\mathrm{LSB}(d_i) = \mathrm{LSB}^1(d_i \lll 1)$. Note that the value $(\delta^{\mathrm{c}}_{w,i} * \mathrm{LSB}(K_{17})) \in \{0,1\}$ specifically deals with the carry bit generated at the LSB-position.

We can evaluate the bit δ_i by *partial* decryption. Since we are rather interested in the bit $\delta^* = \bigoplus_i \delta_i$, we rewrite δ^* by

$$\delta^* = \delta^*_w \oplus (\delta^{*\mathrm{c}}_w * \mathrm{LSB}(K_{17})) \oplus \delta^*_x \oplus \delta^*_z$$

with

$$\begin{aligned}
\delta^*_w &= \bigoplus_{0 \le i < 2^{127}} \delta_{w,i} = \bigoplus_i (\mathrm{LSB}^1(G_1(w_i \oplus K_{18}))), \\
\delta^{*\mathrm{c}}_w &= \bigoplus_{0 \le i < 2^{127}} \delta^{\mathrm{c}}_{w,i} = \bigoplus_i (\mathrm{LSB}(G_1(w_i \oplus K_{18}))), \\
\delta^*_x &= \bigoplus_{0 \le i < 2^{127}} \delta_{x,i} = \bigoplus_i \mathrm{LSB}(G_2(x_i \oplus K_{19})), \\
\delta^*_z &= \bigoplus_{0 \le i < 2^{127}} \delta_{z,i} = \bigoplus_i (\mathrm{LSB}^1(z_i) \oplus \mathrm{LSB}^1(K_{21})) = \bigoplus_i \mathrm{LSB}^1(z_i).
\end{aligned}$$

Assume an adversary to know (or to have guessed) the S-boxes, i.e., to know the functions G_1 and G_2. Given 2^{127} ciphertexts (w_i, x_i, y_i, z_i), we can compute the bits δ^*_w, $\delta^{*\mathrm{c}}_w$, δ^*_x and δ^*_z independently.

For δ^*_z just count how often one of the 2^{127} bits $\mathrm{LSB}^1(z_i)$ is one – δ^*_z is just this number (modulo 2). Regarding δ^*_x we just need to consider all words x_i which appear an odd time as a part of a ciphertext (w_k, x_i, y_k, z_k). These are at most 2^{32} words x_i. Given these and the key word K_{19} we just count mod 2 how often $\mathrm{LSB}(G_2(x_i \oplus K_{19}))$ is one. Computing δ^*_w and $\delta^{*\mathrm{c}}$ can be done similarly.

4.2 Attacking Seven Rounds with Full Whitening

In this section we describe and analyse the seven-round attack. Consider the following algorithm.

0. Choose 2^{127} plaintexts as required for the six-round distinguisher. Ask for the corresponding ciphertexts (w_i, x_i, y_i, z_i), $0 \le i < 2^{127}$.
1. For all $W \in \{0,1\}^{32}$: count (mod 2) the ciphertexts (W, x_i, y_i, z_i).
2. For all $X \in \{0,1\}^{32}$: count (mod 2) the ciphertexts (w_i, X, y_i, z_i).

3. Evaluate δ_z^* by counting the bits $\mathrm{LSB}^1(z_i)$ of the words z_i.
4. Guess the key-dependent vector S, defining the functions G_1 and G_2.
5. Consider the 2^{32} one-bit counters for ciphertexts (w_i, X, y_i, z_i). For every key $K_{19} \in \{0,1\}^{32}$ evaluate δ_x^*. Write $\delta_x^*(K_{19})$ for the results.
6. Consider the 2^{32} one-bit counters for ciphertexts (W, x_i, y_i, z_i). For every choice of K_{18} evaluate δ_w^* and $\delta_w^{*\mathrm{c}}$. Write $\delta_w^*(K_{18})$ and $\delta_w^{*\mathrm{c}}(K_{18})$ for the results.
7. Check the subkey triple $(\mathrm{LSB}(K_{17}), K_{18}, K_{19}) \in \{0,1\} \times \{0,1\}^{32} \times \{0,1\}^{32}$. It is is *"valid for S"* if and only if

$$\delta_w^*(K_{18}) \oplus (\delta_w^{*\mathrm{c}}(K_{18}) * \mathrm{LSB}(K_{17})) \oplus \delta_x^*(K_{19}) = \delta_z^*.$$

How expensive is this algorithm? Let $K \in \{128, 192, 256\}$ be the key size, i.e. each of the "key halfs" M_e, M_o, and S consists of $K/64 \in \{2, 3, 4\}$ words. Obviously, we need 2^{127} chosen plaintexts, half of the codebook. Each of the steps 1–3 requires 2^{127} very simple operations (incrementing a one-bit counter). Step 4 requires to guess $K/2$ key bits. Each of the steps 5–7 is repeated $2^{K/2}$ times:

- Note that we can compute and store a table with 2^{32} entries for the function G_1. Then for each key K_{19}, step 5 requires an average of 2^{31} very simple operations (essentially table look-ups).
- Since $G_2(x) = G_1(x \ggg 8)$, we can reuse the table for G_1 for the function G_2. Then for Each key K_{18}, step 6 requires $2 * 2^{32}$ very simple operations, on the average.

Hence this algorithm requires about $3 * 2^{(K/2)+64}$ *"very simple"* operations. Since such an operation is much faster than a single Twofish encryption, the attack is much faster than exhaustive key search for $K \geq 128$.

The algorithm allows us to filter out half of the keys. (I.e., about 50% of all random permutations pass the test.) So a key finding attack based on the algorithm has to exhaustively search the remaining half of the key space, requiring an expected number of 2^{K-2} Twofish encryptions.

4.3 Attacking Eight Rounds without Post-whitening

The previous attack can be modified to work for eight rounds of Twofish without post-whitening. Let the usual 2^{127} plaintext be chosen. After six rounds, including the swap, we have 2^{127} quadruples (a_i, b_i, c_i, d_i) of 32 bit words, and

$$\delta^* = 0 = \bigoplus_{0 \leq i < 2^{127}} \mathrm{LSB}(d_i).$$

Applying the seven round functions changes these to (a_i, b_i, e_i, f_i), the swap to (e_i, f_i, a_i, b_i), and the last round to (e_i, f_i, g_i, h_i). See Figure 6. Note that K_{18} and K_{19} are no longer part of the post-whitening key, but constitute the round key for round eight.

Set $x_i = G_1(e_i) + G_2(f_i)$ and $y_i = G_1(e_i) + 2G_2(f_i)$. For every key-dependent k-word vector S we know a mapping from the 2^{64} pairs (e_i, f_i) to the pairs

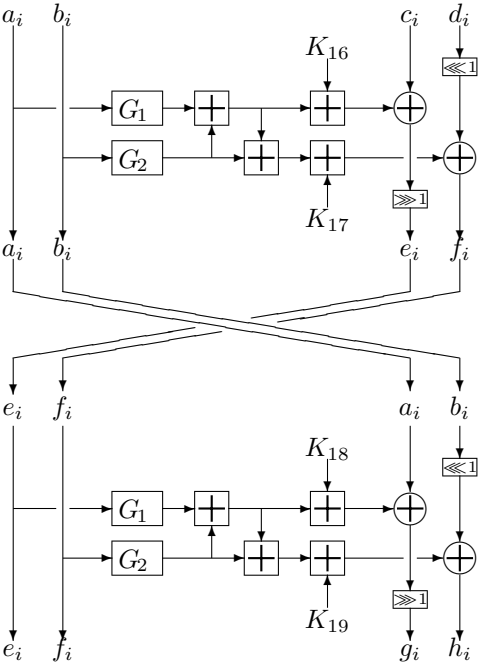


Fig. 6. Round Seven and Eight Without Post-Whitening.

(x_i, y_i). So we can virtually blow up the 2^{127} ciphertext quadruples (e_i, f_i, g_i, h_i) to six-tuples $(e_i, f_i, x_i, y_i, g_i, h_i)$. Computing the mapping requires about the time of 2^{64} one-round decryptions.

The attack on eight rounds of Twofish works quite similarly to the previous attack. We decompose σ^* into key-dependent bits $\delta_a^*(K_{18})$, $\delta_a^{*c}(K_{18})$, $\delta_b^*(K_{19})$ and δ_f with

$$\delta_a^*(K_{18}) = \bigoplus_i \left(\mathrm{LSB}^1(G_1(a_i))\right) = \bigoplus_i \left(\mathrm{LSB}^1(G_1((x_i + K_{18}) \oplus (g_i \lll 1)))\right),$$

$$\delta_a^{*c}(K_{18}) = \bigoplus_i \left(\mathrm{LSB}(G_1(a_i))\right) = \bigoplus_i \left(\mathrm{LSB}(G_1((x_i + K_{18}) \oplus (g_i \lll 1)))\right),$$

$$\delta_b^*(K_{19}) = \bigoplus_i \left(\mathrm{LSB}(G_2(b_i))\right) = \bigoplus_i \left(\mathrm{LSB}(G_2(((y_i + K_{19}) \oplus h_i) \ggg 1))\right),$$

$$\text{and} \quad \delta_f^* = \bigoplus_i \left(\mathrm{LSB}^1(f_i)\right).$$

As above, a subkey triple $(\mathrm{LSB}(K_{17}), K_{18}, K_{19}) \in \{0,1\} \times \{0,1\}^{32} \times \{0,1\}^{32}$ is *"valid for S"* if and only if

$$\delta_a^*(K_{18}) \oplus (\delta_a^{*c}(K_{18}) * \mathrm{LSB}(K_{17})) \oplus \delta_b^*(K_{19}) = \delta_f^*.$$

The basic idea for the attack is to run a filtering algorithm to sort out 50% of the keys, and then to exhaustively search the remaining half of the keys. First we guess the key-dependent vector S, defining G_1 and G_2. When G_1 and G_2 are known, computing $\delta_b^*(K_{19})$ takes roughly 2^{64} very simple steps for each key K_{19}. Similarly, computing $\delta_a(K_{18})$ and $\delta_a^{*c}(K_{18})$ takes roughly 2^{64} very simple steps for each key K_{18}. The filtering is significantly faster than exhaustive key search for $K \geq 192$, key finding using this attack takes the equivalent of about 2^{K-2} encryptions and hence is twice as fast as exhaustively searching the entire key space.

4.4 An Improvement for 256-Bit Keys

Reconsider seven rounds of Twofish and 2^{127} plaintexts chosen as before, as described in Section 4.2 and Figure 5. So far, we only have used one distinguishing property: the bits at the LSB-position of the words d_i are balanced. Thus we could filter out half of the keys. Using the second property too, we can filter out 75 % of the keys: the bits at the MSB-position of the c_i-words are balanced.

Set $u_i = G_1(w_i \oplus K_{18}) + G_2(x_i \oplus K_{19})$ and $v_i = u_i \oplus K_{16}$. Then $c_i = v_i \oplus ((y_i \oplus K_{20}) \ggg 1)$. The bits at the MSB-position of c_i are balanced, i,e.,

$$\bigoplus_{0 \leq i < 2^{127}} \mathrm{MSB}(c_i) = 0 \quad \Longleftrightarrow \quad \bigoplus_{0 \leq i < 2^{127}} \mathrm{MSB}(v_i) = \bigoplus_{0 \leq i < 2^{127}} \mathrm{MSB}(y_i \lll 1).$$

Consider running the following algorithm:

0. Choose 2^{127} plaintexts as before and ask for the ciphertexts (w_i, x_i, y_i, z_i).
1. For all $(W, X) \in (\{0,1\}^{32})^2$: count (mod 2^{32}) the ciphertexts (W, X, y_i, z_i).
2. Evaluate δ_y^{**} by counting the bits $\mathrm{MSB}(y_i \lll 1)$
3. Guess the key-dependent vector S, defining the functions G_1 and G_2.
4. For each of the 2^{64} key pairs (K_{18}, K_{19}), and every $u_i \in \{0,1\}^{32}$:
   Use the 2^{64} counters from Step 1 to compute counters $\mathrm{cnt}[u_i, K_{18}, K_{19}]$ for the values $u_i = G_1(w_i \oplus K_{18}) + G_2(x_i \oplus K_{19})$.
5. For each of the 2^{32} keys K_{16}, compute

$$\delta_v^{**}(K_{18}, K_{18}, K_{20}) = \bigoplus_i \mathrm{MSB}((u_i + K_{16}) * \mathrm{cnt}[u_i, K_{18}, K_{19}]).$$

6. The triple $(K_{16}, K_{18}, K_{19}) \in (\{0,1\}^{32})^3$ is *"good for S"* if and only if

$$\delta_v^{**}(K_{18}, K_{18}, K_{20}) = \delta_y^{**}.$$

Similar to Section 4.2, the above algorithm requires 2^{127} chosen plaintexts, and each of the steps 1–2 requires 2^{127} very simple operations (incrementing a 32-bit counter). Step 4 requires to guess $K/2$ key bits, and both the steps 4 and 5 are repeated $2^{K/2}$ times. Step 4 deals with 2^{64} counters and 2 key words, and step 5 deals with 2^{32} values u_i and 3 key words. Hence, both steps require 2^{128} very simple operations.

The entire algorithm requires about $2 * 2^{(K/2)+64+64} = 2^{(K/2)+129}$ *"very simple"* operations and is clearly faster than exhaustive key search for $K \geq 256$.

The same technique works for the attack on eight rounds of Twofish without post-whitening, too. So for $K = 256$ a key finding attack based on checking subkeys being both *valid* and *good* requires computation time equivalent to about 2^{K-3} Twofish encryptions.

4.5 Finding the Key: Summary

In Table 2 we summarise our key finding attacks and compare them with exhaustive key search. Note that the running time depends on the number $K \in \{128, 192, 256\}$ of key bits, and we use one encryption as unit of time.

Table 2. Key Finding Attacks for Twofish.

Attack	Plaintexts	Rounds	Key Bits	Whitening		Running Time
				Pre-	Post-	
Section 4.2	2^{127} chosen	7	$K \geq 128$	yes	yes	2^{K-2}
Section 4.2 / 4.4	2^{127} chosen	7	$K = 256$	yes	yes	2^{K-3}
Section 4.3	2^{127} chosen	8	$K \geq 192$	yes	no	2^{K-2}
Section 4.3 / 4.4	2^{127} chosen	8	$K = 256$	yes	no	2^{K-3}
ex. key search	≈ 3 known	any	any	any	any	2^{K-1}

Note that our attacks are only 2–4 times faster than exhaustive key search, because we filter out 50 % or 75 % of the keys. The overwhelming part of the running time is needed to exhaustively search the remaining key space. Improved distinguishers for six rounds of Twofish, filtering out more keys, could speed up our attacks. Finding such improved distinguishers remains an open problem.

5 Conclusion

At present, this paper describes the best known attack on the AES finalist Twofish. We can break up to 8 rounds of Twofish with one-sided whitening faster than exhaustively searching the key would require, though our attacks are only 2–4 times faster than exhausive key search. Since Twofish is a 16-round cipher with twosided whitening, we are able to penetrate exactly one half of Twofish. This still leaves Twofish with a reasonable security margin. An interesting side-note is that the one-bit rotates in Twofish appear to be useful – a variant of without the rotates would allow some enhancement of our attacks.

Also, the paper demonstrates the usefulness of saturation attacks for attacking ciphers which are not square-like. It would be interesting to use saturation techniques to attack other cryptographic primitives.

Acknowledgements. Supported by Counterpane and Hi/fn, the author participated at the "Third Twofish Retreat" in Utah, USA. He thanks the workshop participants for their encouragement in considering saturation attacks for Twofish. Also, the FSE referees provided useful comments to improve this presentation.

References

1. E. Biham, A. Biryukov, A. Shamir, "Miss in the Middle Attacks on IDEA and Khufru", *Fast Software Encryption 1999*, Springer LNCS 1636, pp. 124–138.
2. E. Biham, "Cryptanalysis of Ladder DES", *Fast Software Encryption 1997*, Springer LNCS 1267, pp. 134–138.
3. C. D'Halluin, G. Bijnens, V. Rijmen, B., Preneel, "Attack on six round of Crypton", *Fast Software Encryption 1999*, Springer LNCS 1636, pp. 46–59.
4. J. Daemen, L. Knudsen, V. Rijmen: "The block cipher Square", *Fast Software Encryption 1997*, Springer LNCS 1267, pp. 149–165.
5. J. Daemen, V. Rijmen, "AES proposal: Rijndael" (2nd version) [18].
6. N. Ferguson, "Impossible differentials in Twofish", Twofish TR #5, 1999 [18].
7. N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, D. Whiting, "Improved Cryptanalysis of Rijndael" *Fast Software Encryption 2000*, to appear.
8. J. Kelsey, "Key Separation in Twofish", Twofish TR #7, 2000 [18].
9. L. Knudsen, "Truncated and Higher Order Differentials", *Fast Software Encryption 1995*, Springer LNCS 1008, pp. 196-211.
10. L. Knudsen, "Trawling Twofish (revisited)" May 15, 2000, [18].
11. C. H. Lim, "Crypton: a new 128-bit block cipher" [18].
12. C. H. Lim, "A revised version of Crypton – Crypton V 1.0 – ", Fast Software Encryption 1999, Springer LNCS 1636, pp. 31–45.
13. F. Mirza, S. Murphy, "An Observation on the Key Schedule of Twofish", In: *The 2nd Advanced Encryption Standard Conference*, pp. 151–154, April 1999 [18].
14. S. Murphy, "The Key Separation of Twofish", 2000 [18].
15. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, "Twofish: A 128-bit Block Cipher" [18].
16. B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson, *"The Twofish Encryption Algorithm"*, Wiley, 1999.
17. "The GNU Privacy Guard", "`http://www.gnupg.org/`".
18. "AES Development Effort", NIST, "`http://www.nist.gov/aes`".

Linear Cryptanalysis of Reduced Round Serpent*

Eli Biham[1], Orr Dunkelman[1], and Nathan Keller[2]

[1] Computer Science Department, Technion – Israel Institute of Technology, Haifa 32000, Israel,
{biham,orrd}@cs.technion.ac.il, http://www.cs.technion.ac.il/~biham/.
http://vipe.technion.ac.il/~orrd/me/.
[2] Mathematics Department, Technion – Israel Institute of Technology, Haifa 32000, Israel,
nkeller@tx.technion.ac.il

Abstract. Serpent is one of the 5 AES finalists. In this paper we present a 9-round linear approximation for Serpent with probability of $1/2+2^{-52}$. We use it to attack 10-round Serpent with all key lengths with data complexity of 2^{118} and running time of 2^{89}. A variant of this approximation is used in the first attack against an 11-round Serpent with 192-bit and 256-bit keys, which require the same amount of data and 2^{187} running time.

1 Introduction

Serpent [1] is a block cipher which was suggested as a candidate for the Advanced Encryption Standard (AES) [9], and was selected to be among the five finalists.

In [5] a modified variant of Serpent in which the linear transformation was modified into a permutation was analyzed. The permutation allows one active S box to activate only one S box in the consecutive round, a property that cannot occur in Serpent. Thus, it is not surprising that this variant is much weaker than Serpent, and that it can be attacked with up to 35 rounds.

In [7] the 256-bit variant of Serpent up to 9 rounds is attacked using an amplified boomerang attack. The attack is based on building a 7-round distinguisher for Serpent, and using it for attacking up to 9 rounds. The distinguisher is built using the amplified boomerang technique. It uses a 4-round differential characteristics in rounds 1–4, and a 3-round characteristic in rounds 5–7.

In [4] 10-round Serpent with 256-bit keys was analyzed using the rectangle attack. Also the best known 3-round, 4-round, 5-round and 6-round differential characteristics of Serpent have been presented.

In this paper we present the best known linear approximation of Serpent of 9 rounds which has a probability of $1/2 + 2^{-52}$. We use a variant of this approximation (with bias of 2^{-58} in order to attack 10-round Serpent with all key lengths with data complexity of 2^{118} and running time of 2^{89} memory accesses.

* This work was supported by the European Union fund IST-1999-12324 – NESSIE

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 16–27, 2002.

Using another variant of this approximation allows us to attack up to 11-round Serpent with 192-bit and 256-bit keys, using the same amount of data and which requires time equivalent to 2^{187} 11-round Serpent encryptions.

The paper is organized as follows: In Section 2 we give the description of Serpent. In Section 3 we present the 9-round linear approximation of Serpent. In Section 4 we describe the search methodology we used to find the approximation. In Section 5 we present the linear attack on 10-round Serpent with all possible key lengths. In Section 6 we present the linear attack on 11-round Serpent with 256-bit keys. Section 7 summarizes the paper.

2 A Description of Serpent

Serpent [1] is a SP-network block cipher with block size of 128 bits and 0–256 bit keys. It is composed of alternating layers of key mixing, S boxes and linear transformation. We deal only with the equivalent bitsliced description.

The key scheduling algorithm of serpent accepts 128-bit, 192-bit and 256-bit keys. Keys shorter than 256 bits are padded by 1 followed by as many 0's needed to have a total length of 256 bits. The key is then used to derive 33 subkeys of 128 bits.

We use the notations of [1]. Each intermediate value of round i is denoted by $\hat{B}_i$ (a 128-bit value). The 32 rounds are numbered 0–31. Each $\hat{B}_i$ is treated as four 32-bit words X_0, X_1, X_2, X_3, where bit j of X_i is bit $4 * i + j$ of the 128 bit value $\hat{B}_i$ (i.e., bit j of $X_3, \ldots, X_0$ is the j'th nibble, with the bit from X_3 as the most significant bit).

Serpent has a set of eight 4-bit to 4-bit S boxes. Each round function R_i ($i \in \{0, \ldots, 31\}$) uses a single S box 32 times in parallel. For example, R_0 uses S_0, 32 copies of which are applied in parallel. Each S box permute nibble, i.e., for $0 \leq j \leq 31$ we take the j'th bit of $X_3, \ldots, X_0$ and return the value according to the S box.

The set of eight S-boxes is used four times. In round i we use $S_{i \bmod 8}$, i.e., S_0 is used in round 0, etc. The last round (round 31) is slightly different from the others: apply S_7 on $\hat{B}_{31} \oplus \hat{K}_{31}$, and XOR the result with $\hat{K}_{32}$ rather than applying the linear transformation.

The cipher may be formally described by the following equations:

$$\begin{aligned} \hat{B}_0 &:= P \\ \hat{B}_{i+1} &:= R_i(\hat{B}_i) \\ C &:= \hat{B}_{32} \end{aligned}$$

where

$$\begin{aligned} R_i(X) &= LT(\hat{\mathcal{S}}_i(X \oplus \hat{K}_i)) \qquad i = 0, \ldots, 30 \\ R_i(X) &= \hat{\mathcal{S}}_i(X \oplus \hat{K}_i) \oplus \hat{K}_{32} \quad i = 31 \end{aligned}$$

where $\hat{\mathcal{S}}_i$ is the application of the S-box $S_{i \bmod 8}$ 32 times in parallel, and LT is the linear transformation.

The linear transformation: The 32 bits in each of the output words are linearly mixed by

$$\begin{aligned}
X_0, X_1, X_2, X_3 &:= \mathcal{S}_i(B_i \oplus K_i) \\
X_0 &:= X_0 <<< 13 \\
X_2 &:= X_2 <<< 3 \\
X_1 &:= X_1 \oplus X_0 \oplus X_2 \\
X_3 &:= X_3 \oplus X_2 \oplus (X_0 << 3) \\
X_1 &:= X_1 <<< 1 \\
X_3 &:= X_3 <<< 7 \\
X_0 &:= X_0 \oplus X_1 \oplus X_3 \\
X_2 &:= X_2 \oplus X_3 \oplus (X_1 << 7) \\
X_0 &:= X_0 <<< 5 \\
X_2 &:= X_2 <<< 22 \\
B_{i+1} &:= X_0, X_1, X_2, X_3
\end{aligned}$$

where $<<<$ denotes rotation, and $<<$ denotes shift.

In the last round, this linear transformation is replaced by an additional key mixing: $B_{32} := S_7(B_{31} \oplus K_{31}) \oplus K_{32}$.

3 A 9-Round Linear Approximation

We adopt the representation similar to the differential characteristics as in [6,4], and add more data to the figures. In our representation, the rows are the bitsliced 32-bit words, and each column is the input to a different S box. The first line represents X_0, the last line represents X_3, and the rightmost column represents the least significant bit of the words. A thin arrow represents an approximation with probability $\frac{1}{2} \pm \frac{1}{8}$ for the specific S box (given the input parity, the output parity is achieved with probability $\frac{1}{2} \pm \frac{1}{8}$), and a fat arrow stands for probability $\frac{1}{2} \pm \frac{1}{4}$. If the bit is taken into consideration of the parity, the box related to it is filled. Example for our notation can be found in Figure 1, in which in the first S box (related to bits 0) the parity of 1 equals to the parity of 3 in the output with probability $1/2 \pm 1/4$ (1/4 or 3/4), and in S box 30 the parity of the input 3 causes an output parity of 1 with probability $1/2 \pm 1/8$ (3/8 or 5/8).

This 9-round linear approximation has a probability of $1/2 + 2^{-52}$, in round 3 (or 11 or 19 or any other round using S_3) the following approximation holds with bias of 2^{-11}:

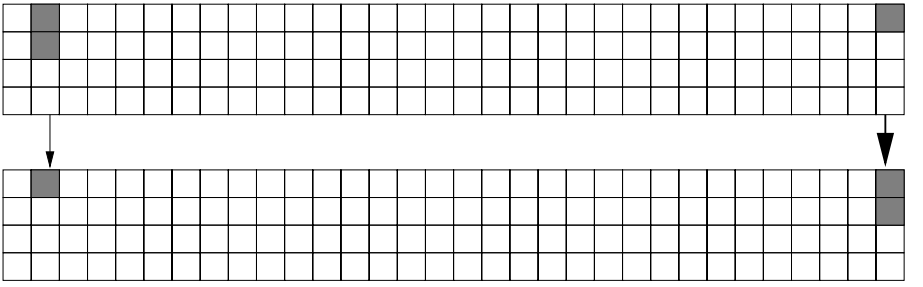
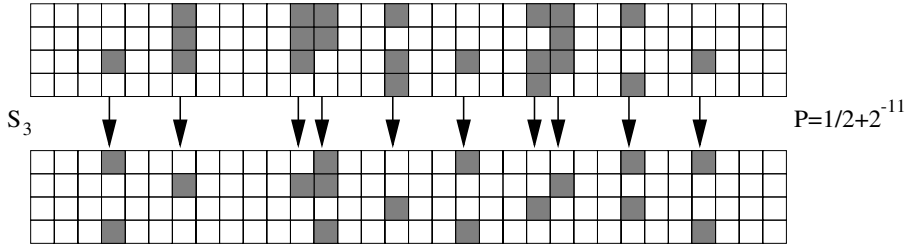
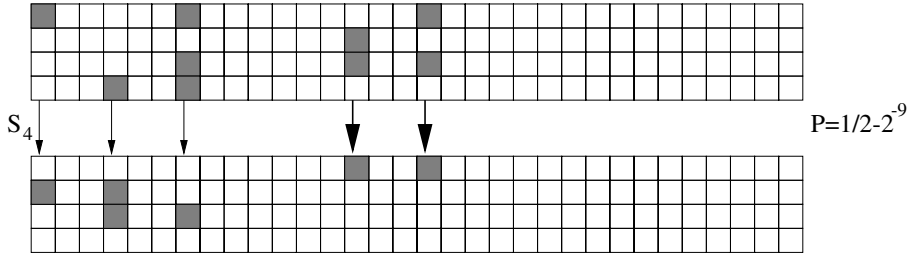


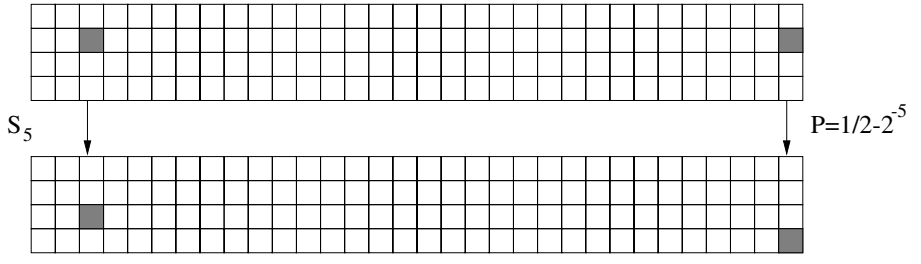
Fig. 1. Linear Approximation Representation Example



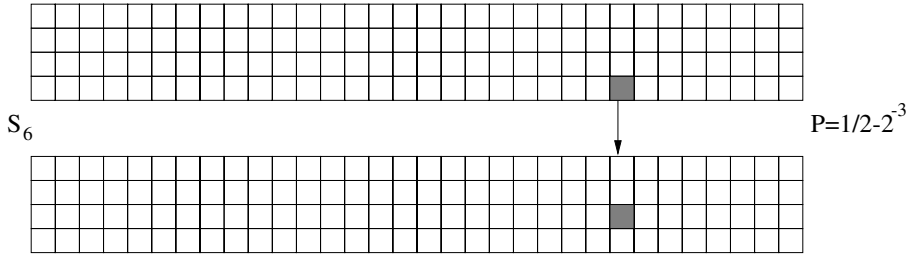
After the linear transformation, and the application of S_4 we get the following linear approximation with bias of 2^{-9}:



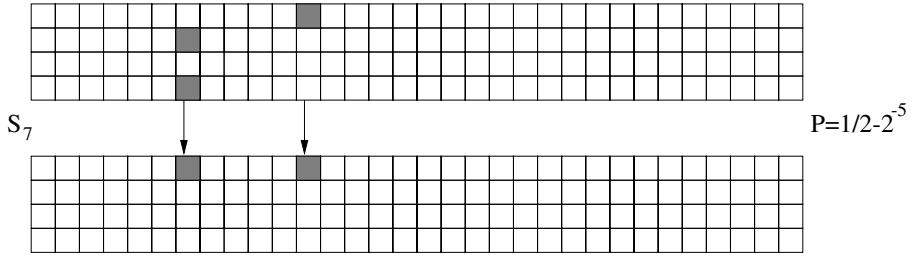
After the linear transformation, and the application of S_5 we get the following linear approximation with bias of 2^{-5}:



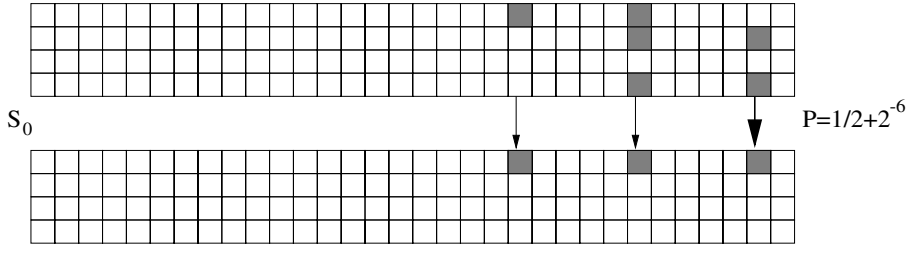
After the linear transformation, and the application of S_6 we get the following linear approximation with bias of 2^{-3}:



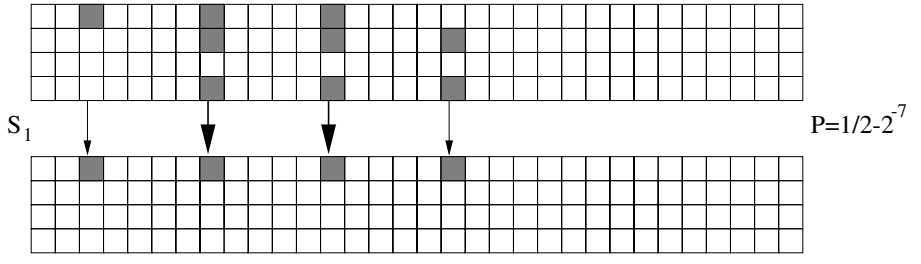
After the linear transformation, and the application of S_7 we get the following linear approximation with bias of 2^{-5}:



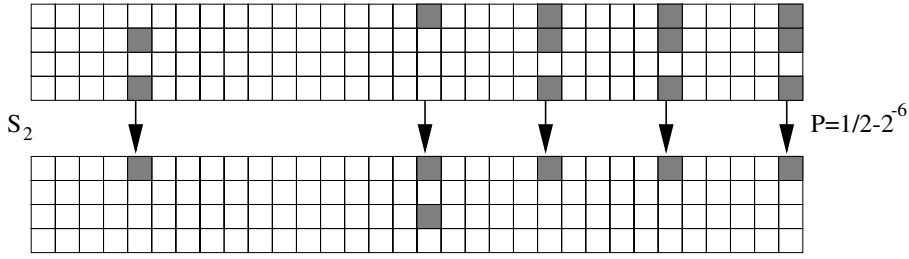
After the linear transformation, and the application of S_0 we get the following linear approximation with bias of 2^{-6}:



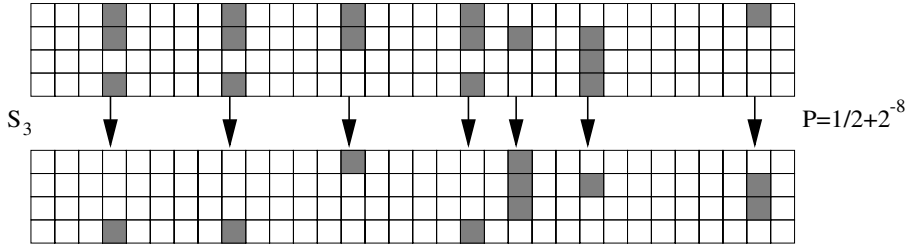
After the linear transformation, and the application of S_1 we get the following linear approximation with bias of 2^{-7}:



After the linear transformation, and the application of S_2 we get the following linear approximation with bias of 2^{-6}:



After the linear transformation, and the application of S_3 we get the following linear approximation with bias of 2^{-8}:



The total bias of the approximation is therefore $2^8 \cdot 2^{-11} \cdot 2^{-9} \cdot 2^{-5} \cdot 2^{-3} \cdot 2^{-5} \cdot 2^{-6} \cdot 2^{-7} \cdot 2^{-6} \cdot 2^{-8} = 2^{-52}$.

We can play with the input subsets and output subsets. There are 2^{10} possible input biases (all with the same bias, half of them has probability of $1/2 + 2^{-11}$ and half with probability $1/2 - 2^{-11}$), and 2^7 output subsets with the same bias. Thus, receiving 2^{17} possible approximations with the same bias. 2^{16} have probability $1/2 + 2^{-52}$, and the other 2^{16} have probability $1/2 - 2^{-52}$.

These approximations can also be rotated by one or two bits to the left, due to the nature of the linear transformation. Thus, we have in total $3 \cdot 2^{17}$ approximations.

4 Search Methodology

We have searched for the the linear characteristic in the following way:

- We searched over all one bit inputs to the linear transformation and looked for the cases where the minimal number of S boxes are affected in the next round.
- We searched over all one bit outputs from the linear transformation and looked for the cases where the minimal number of S boxes are affected in the previous round.
- We observed that knowing a least significant bit of a nibble (bit which is in X_0) in the entrance to the linear transformation, affects 2 to 3 S boxes in the next round.
- We observed that knowing a most significant bit of a nibble (bit which is in X_3) in the exit from the linear transformation, affects 2 to 3 S boxes in the previous round.

- For S box 7, 8 and 9 (out of the parallel 32), the input has only two active S boxes in the round before, and the output cause only two active S boxes in the next round. Thus, we found a 3-round approximation with 5 active S boxes.
- We tried to add more rounds to the approximation, using the fact that when we go backward to the previous round, it is better to take subsets of bits from X_1 and X_3, and while going forward it is better to take subsets of bits from X_0 and X_2.
- By iteratively adding rounds in the beginning and in the end of the approximation we have found a 9-round approximation.

The approximations' details we found in the way on fewer rounds are given in Table 1. The approximations themselves can be easily derived from the 9-round approximation by removing the unnecessary rounds, and making the edge rounds (first and last round) of the approximation with maximal bias. From this table it becomes clear that the authors of Serpent made a mistake in their claimed bounds, as the biases found here are 4-8 times higher than the bounds in [1]. However, this mistake does not affect the security claims for the whole cipher, as there is a huge distance between a 9-round approximation and attacking 32 rounds, or even 16 rounds of Serpent.

Table 1. Linear Approximations for Serpent

Number of Rounds	Starting from	Number of Active S boxes	Bias
3	S_5	5	2^{-7}
4	S_5	8	2^{-12}
5	S_5	12	2^{-18}
6	S_4	17	2^{-25}
6	S_5	17	2^{-25}
7	S_4	22	2^{-32}
8	S_4	29	2^{-39}
9	S_3	39	2^{-52}

5 Attacking 10-Round Serpent

We can attack 10-round Serpent (in rounds 3–12) using a 9-round linear approximation (in rounds 3–11), which is similar to the one presented in Section 3, besides the last round of the approximation (round 11) which is shown in Figure 2. This replacement gives us a linear approximation with probability of $1/2 - 2^{-58}$ (instead of $1/2 + 2^{-52}$), but we reduce by 12 the number of active S boxes in round 12, as this last round activates only 11 S boxes in round 12 instead of 23 in the original approximation.

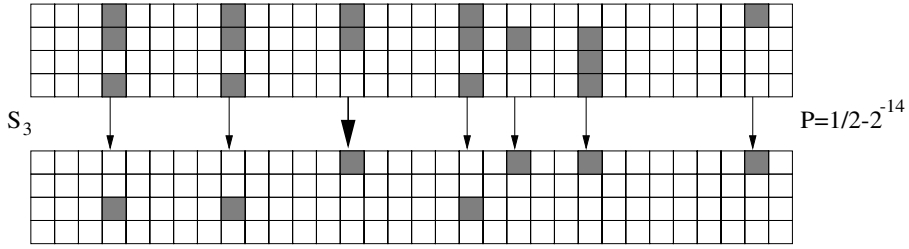


Fig. 2. Last Round of the 9-round Linear Approximation used in the Attack

The basic algorithm of the attack is based on the basic M2 algorithm [8]. Given 2^{118} plaintexts perform the following algorithm:

- Initialize array of 2^{44} counters (counters must be at least 118 bits long).
- For each of the 44-bit subkey value of the 11 S boxes in the last round:
  - Decrypt each ciphertext using the subkey value in the 11 S boxes in the last round.
  - Calculate the input subset parity and the output subset parity required by the approximation, and calculate the XOR of two. If the XOR value is zero increase the counter related to the subkey by 1, otherwise decrease by 1.
- look over the counters, and choose the one with the maximal absolute value.

It is expected that the maximal deviation would be suggested for the right subkey. Moreover, we can take into consideration several subkeys with the highest Absolute values, and increase the success rate (on the expense of running time).

We can use other approximations involving as few as possible S boxes with unknown subkeys (there are many such approximations, even with one new S box), and receive additional bits of subkey material. This way one can find as many as 112 bits of the last round subkey. We now decrypt one round (guessing 16 bits from the last round subkey) and perform attack on 9-round Serpent (which is much faster than attacking 10-round Serpent). Due to time complexity this attack is applicable only on the 192-bit and 256-bit keys variants.

The time complexity for this algorithm is the time needed to decrypt 11 S boxes (out of 320) under 2^{44} keys for 2^{118} ciphertexts. Thus, the time complexity is $2^{118} \cdot 2^{44} \cdot 11/320 \approx 2^{157.14}$ 10-round Serpent encryptions, with data complexity of 2^{118} plaintexts, and memory of 2^{44} counters.

Note that for each ciphertext we look only on 44 bits, and decrypt many times the same value under the same key. These give rise to a better algorithm:

- Initialize array of 2^{45} counters (counters must be at least 118 bits long).
- For each plaintext P calculate the input subset parity, and advance the counter which is related to the parity and the 44 bits from the output of 11 S boxes in the last round.
- Initialize array of 2^{44} counters (counters must be at least 118 bits long).

- For each value of the subkey do the following:
  - Decrypt each possible 44 bits (of ciphertext) and calculate the output subset parity. Increase/decrease the related counter related to subkey according the XOR of the parities (the parity from the first table, and the calculated parity) by the number of plaintext/ciphertext pairs satisfying the parity and the 44 bits of ciphertext.
- look over the counters, and choose the one with the maximal absolute value.

This way, each 44-bit ciphertext value is decrypted only twice under each subkey, and we note that the variant of the algorithm which uses only one decryption is trivial.

Thus, the time complexity of the attack is the time required to perform $2^{45} \cdot 2^{44} = 2^{89}$ decryptions of 11 S boxes (as the first step is just counting the number of occurrences of a bit string), which is equivalent to $2^{89} \cdot 11/320 = 2^{84.13}$ 10-round Serpent encryptions. Again we can use other approximations (there are another 15 approximations with 11 active S boxes in the last round), and again retrieve 112 bits from the last round subkey. For 128-bit keys we can now just guess the remaining 16 bits, and for longer keys we just guess those 16 bits, and decrypt one round to attack 9-round Serpent.

We can also use other linear approximations. There is a linear approximation with bias 2^{-57}, with only 12 active S boxes in the last round. Using this approximation instead, would result in 2^{116} plaintexts needed, with 2^{49} counters, and $2^{49} \cdot 2^{48} = 2^{97}$ times we decrypt 12 S boxes, which is equivalent to $2^{97} \cdot 12/320 = 2^{92.26}$ 10-round Serpent encryptions.

For 192-bit and 256-bit keys we can use the original 9-round linear approximation, counting on 23 S boxes and reducing the required plaintexts to 2^{106}. The memory complexity would be in such a case 2^{93} counters, and the time needed is equivalent to $2^{93} \cdot 2^{92} = 2^{185}$ times decrypting 23 S boxes.

6 Attacking 11-Round 256-Bit Key Serpent

We can attack 11-round Serpent (in rounds 2–12) using a 9-round linear approximation (in rounds 3–11), which is similar to the one presented in Section 5 (the modified approximation with probability $1/2 + 2^{-58}$), besides the first round which is shown in Figure 3. This replacement gives us approximation with bias of 2^{-58} and we reduce by 5 the number of active S boxes in round 2, as this first round of the approximation has only 24 active S boxes in round 2 (instead of 29 in the original approximation) and 11 active S boxes in the last round (instead of 23 in the original approximation).

The basic algorithm in this attack is quite similar to the algorithm from Section 5. Given 2^{118} plaintexts:

- Initialize array of $2^{44+96} = 2^{140}$ counters (counters must be at least 118 bits long).
- For each of the 44-bit subkey value in the 11 S boxes in the last round and 96-bit subkey value in the 24 S boxes in the first round:

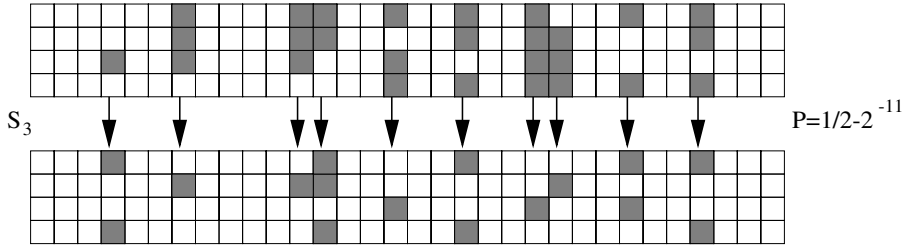


Fig. 3. First Round of the 9-round Linear Approximation used in the 11-round Attack

  - Decrypt each ciphertext through the 11 S boxes in round 12.
  - Encrypt each plaintext through the 24 S boxes in round 2.
  - Calculate the input subset parity and the output subset parity required by the approximation, and calculate the XOR of two. If the XOR value is zero increase the counter related to the subkey by 1, otherwise decrease by 1.
- look over the counters, and choose the one with the maximal absolute value.

The time complexity for this algorithm would therefore be the time needed to encrypt/decrypt 35 S boxes (out of 352) under 2^{140} keys for 2^{118} plaintexts/ciphertexts. Thus, the time complexity is $2^{118} \cdot 2^{140} \cdot 35/352 \approx 2^{255}$ 11-round Serpent encryptions, with data complexity of 2^{118} plaintexts, and memory needed for 2^{140} counters. As one can see this attack is marginally faster than exhaustive search.

We can also use the better algorithm from the previous section. This time we will count according the 96 bits from the plaintext, and take the output subset parity. However, in order to perform the attack, we need to know the parity itself, which we do not know due to the additional round. The solution would be to try for each of the 2^{44} possible subkeys of the last round the improved attack. The algorithm we receive is as follows:

- Initialize array of 2^{44} entries (with entry size of 96+118=214 bits at least)
- For each 44-bit subkey of the last round:
  - decrypt all ciphertext under the subkey, and calculate the output subset parity.
  - Initialize array of 2^{97} counters (counters must be at least 118 bits long).
  - For each plaintext P advance the counter which is related to the output subset parity and the 96 bits from the 24 S boxes.
  - Initialize array of 2^{96} counters (counters must be at least 118 bits long).
  - For each value of the subkey encrypt each possible 96 bits (of plaintext) and calculate the output subset parity. Increase/decrease the related counter related to subkey according the XOR of the parities (the parity from the first table, and the calculated parity) by the number of plaintext/ciphertext pairs satisfying the parity and the 96 bits of plaintext.

  - look over the counters, and choose the one with the maximal absolute value.
  - Keep this 96-bit subkey value for this subkey with the value of the counter in the smaller table.
- Run over the smaller table and look for the entry with the maximal absolute value. The 44-bit subkey related to it, and the 96-bit subkey value stored in the entry has a good chance of being the right ones.

The time complexity for the attack is the time needed to decrypt 2^{118} ciphertexts 11 S boxes under 2^{44} possible subkeys, and for each subkey to run an internal loop which requires $2^{96} \cdot 2^{97} = 2^{193}$ times encrypting 24 S boxes. Thus, the total running time is $2^{44} \cdot (2^{118} \cdot 11/352 + 2^{197}) = 2^{241}$.

Table 2. Known Attacks on Serpent

Rounds	Key Size	Complexity			Source
		Data	Time	Memory	
7	256	2^{122}	2^{248}	2^{126}	[6] - Section 3.5
	all	2^{85}	2^{86} MA	2^{52}	[5] - Section 3
8	192 & 256	2^{128}	2^{163}	2^{133}	[6] - Section 4.2
	192 & 256	2^{110}	2^{175}	2^{115}	[6] - Section 5.3
	256	2^{85}	2^{214} MA	2^{85}	[5] - Section 3.1
9	256	2^{110}	2^{252}	2^{212}	[6] - Section 5.4
10	256	$2^{126.4}$	2^{208}	$2^{131.4}$	[5] - Section 4
	256	$2^{126.4}$	$2^{204.8}$	$2^{195.5}$	[5] - Section 4.1
	all	2^{118}	$2^{89}MA$	2^{45}	This paper
	192 & 256	2^{106}	2^{185}	2^{96}	This paper
11	192 & 256	2^{118}	2^{187}	2^{193}	This paper

MA - Memory Accesses

However, the heavy internal loop (for each of the 2^{97} possible entries and for each 96-bit subkey) can be done only once, and by keeping the result in a large table we reduce the time for each iteration of the external loop to just copy the values in the right place.

Using this better algorithm would require to run once the internal loop (in the cost of 2^{192} times encrypting 24 S boxes, and for each of the 2^{44} last round subkeys to decrypt 2^{118} ciphertext under 11 S boxes, and than access 2^{96} cells in the precomputed table. Thus, the total running time of the attack is equivalent to $2^{192} \cdot 24/352 + 2^{44} \cdot (2^{118} \cdot 11/352 + 2^{96}) = 2^{187}$ 11-round Serpent encryptions. The memory needed for this attack is 2^{193} entries (containing one bit - the input subset parity).

We conclude that this attack is better than exhaustive search for 192-bit and 256-bit keys.

7 Summary

In this paper we performed the first linear cryptanalysis of Serpent, we presented the best known 9-round linear approximation for Serpent with probability $1/2 + 2^{-52}$, and explained our search method.

Using the approximations we have found, we have attacked 10-round Serpent with all bit lengths with time complexity $2^{84.13}$ and 2^{118} known plaintexts. We have also presented an attack on 11-round Serpent with 192-bit and 256-bit keys with time complexity of 2^{187} 11-round Serpent encryptions, with the same data complexity (2^{118}) and 2^{193} memory cells (of one bit).

References

[1] R. Anderson, E. Biham and L. Knudsen, *Serpent: A Proposal for the Advanced Encryption Standard*, NIST AES Proposal 1998.
[2] E. Biham, *A Note on Comparing the AES Candidates*, Second AES Candidate Conference, 1999.
[3] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer-Verlag, 1993.
[4] E. Biham, O. Dunkelman, N. Keller, *The Rectangle Attack - Rectangling the Serpent*, To appear in proceedings of Eurocrypt 2001. Available on-line at *http://vipe.technion.ac.il/ orrd/crypt/*
[5] O. Dunkelman, *An Analysis of Serpent-p and Serpent-p-ns*, rump session, Second AES Candidate Conference, 1999.
[6] T. Kohno, J. Kelsey and B. Schneier, *Preliminary Cryptanalysis of Reduced-Round Serpent*, Third AES Candidate Conference, 2000.
[7] J. Kelsey, T. Kohno and B. Schneier, *Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent*, FSE 7, to appear.
[8] M. Matsui, *Linear Cryptanalysis Method for DES Cipher*, Eurocrypt 93, Springer Verlag LNCS 765, pp. 386-397.
[9] NIST, *A Request for Candidate Algorithm Nominations for the AES*, available on-line at *http://www.nist.gov/aes/*.

Cryptanalysis of the Mercy Block Cipher

Scott R. Fluhrer

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

Abstract. The Mercy block cipher is examined, and a differential attack is shown. Using this differential, a method of distinguishing the cipher from a random permutation is given.

1 Introduction

Mercy is a block cipher that was presented at Fast Software Encryption 2000 [1], and was designed by Paul Crowley. It has a block size of 4096 bits, 128 bits of spice[1], and a variable sized key. It was specifically designed for bulk disk encryption, where each disk block would be encrypted separately, and with the sector index used as the spice. An explicit design goal was that any procedure for distinguishing Mercy encryption using a 128 bit key from a sequence of 2^{128} random permutations should show no more bias toward correctness than a key guessing attack with the same work factor.

We show that this design goal is not achieved, by using a differential to distinguish the cipher from randomness with 2^{27} chosen plaintexts. In addition, we describe how to rederive some of the hidden whitening bits. Further, we analyze why Mercy was vulnerable to this attack.

This paper is structured as follows. In Section 2, the Mercy block cipher is described. Section 3 describes the differential, and section 4 shows how this differential can be used to attack the cipher. Section 5 summarizes our conclusions.

2 Description of Mercy

Mercy is a Feistel network with 6 rounds, and partial pre- and post-whitening. The key is used to derive a substitution table[2] $T : \{0,1\}^8 \rightarrow \{0,1\}^{32}$. The T function is used to define a function $M : \{0,1\}^{32} \rightarrow \{0,1\}^{32}$, as shown on Figure

[1] Spice is a parameter to the cipher, designed to be known to the attacker. It is intended to allow the user to modify the actual permutation implemented by the block cipher, possibly on a per-block basis, without forcing a full rekey

[2] It actually derives affine transformations that is used along with a fixed table $N : \{0,1\}^8 \rightarrow \{0,1\}^8$ to expand the T substitution table, but this is irrelevant to this attack.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 28–36, 2002.

1[3]. The important point for this attack is that, as long as the most significant 8 bits have zero differential, M preserves any differential in the remaining bits with probability 1, shifting that differential over 8 bits.

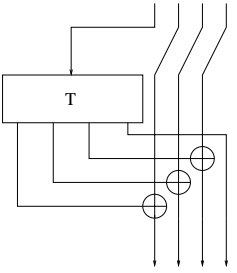


Fig. 1. Operation M, with each line standing for 8 bits.

The M function is used, along with an exclusive-or and two additions modulo 2^{32}, to define the function $Q : \{0,1\}^{128} \times \{0,1\}^{32} \rightarrow \{0,1\}^{128} \times \{0,1\}^{32}$, as shown on Figure 2.

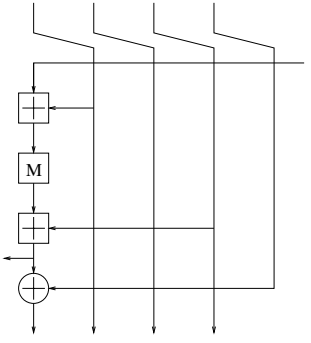


Fig. 2. Operation Q, with each line standing for 32 bits.

The Q function is used to define the function $F : \{0,1\}^{2048} \times \{0,1\}^{128} \rightarrow \{0,1\}^{2048}$, as shown on Figure 3.

The F function is then used as the round function of a six round Feistel network, along with partial pre- and post-whitening, as shown in Figure 4.

There are no attacks on Mercy previous to this attack described in the open literature.

[3] Note that the format of this diagram differs from the diagram in [1] in that the least significant octet on the right.

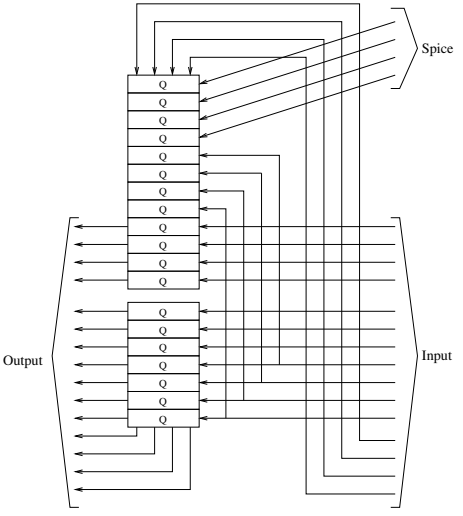


Fig. 3. Operation F, with each line standing for 32 bits.

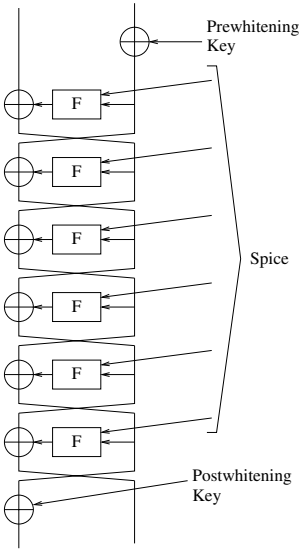


Fig. 4. Mercy, with each line standing for 2048 bits, except for the lines carrying spice, which are 128 bits.

3 Description of the Differential

The first thing to note about this structure is in the F function (Figure 3). If we assume that the last 8 32 bit words of the input halfblock and the spice have

zero differential, then the effect of an input differential[4] in a particular input word can be characterized by a differential in the corresponding output word, and a differential in the 128 descending bits output by that Q box. Normally, a differential in the 128 descending bits would avalanche throughout the remainder of the block. However, what is possible is if several consecutive input words have nonzero differentials, then the descending lines immediately afterwards may have zero differential. If the rest of the input block also has zero differential, that means that the sole effect of that input differential would be an output differential in the corresponding output words.

The second thing to note about Mercy is that, within the Q function, only 8 bits are presented to a key-dependent sbox. All other operations within the Q function are either linear or near linear. Hence, if we arrange things so that the input to the key-dependent sbox always has zero differential, we have a realistic prospect of having a high probability differential.

Figure 5 has the simplest example of such a differential (which is not a differential used in the attack – the differentials actually used are approximately twice as long). A specific difference comes in from the input half-block, flows through the Q boxes as indicated, outputs a specific differential to the output half-block, and leaves the descending bits to the next Q box with a zero differential. The differentials that come in from the input halfblock have been carefully chosen to cancel out the effects of the differentials within the Q box itself before they can avalanche. This is how these types of differentials work.

Also note that this is not a probability 1 differential: in three of the additions (marked by *), we need to assume that the carry between bits[5] 23 and 24 has zero differential, and thus the addition treats the differential exactly the same as an exclusive-or would. Assuming that the actual values of the bits are effectively random (which we will assume), this holds with probability approximately[6] 0.5, and so this differential holds with probability 2^{-3}. The unmarked additions with differential inputs work only with differentials in bit 31. Because the addition ignores the carry from bit 31, addition always treats that differential exactly the same as exclusive-or would.

We also note that the exact place within the halfblocks where the differential occurs doesn't matter. As long as the last 8 32-bit words have zero differential (required because those words are fed into the beginning of the Q stack), the attacker can place the differentials where convenient.

Of course, we need to do more than have a differential for a single round. We need to set up differentials so that it leaves the cipher in a state where the next round differential is also bounded. We need this to occur for at least 4 rounds, for reasons that will be discussed below. It turns out that we can actually manage 5 rounds.

[4] All differentials considered within this paper are bitwise differentials.

[5] This paper will use the convention that the least significant bit in a word is labeled as bit 0.

[6] Because the carry is biased slightly towards zero, the probability is slightly higher.

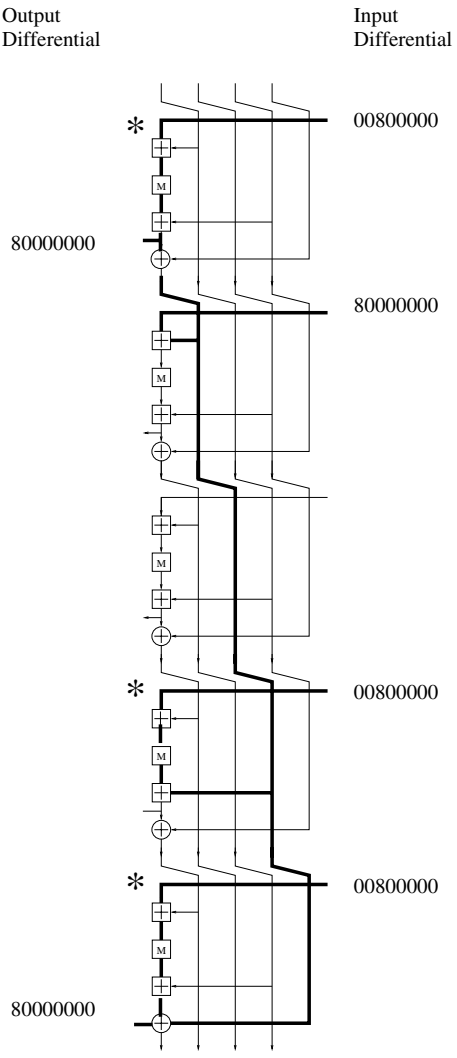


Fig. 5. An example differential within 5 Q boxes contained within the F function. Boldened lines contain a nonzero differential, and the input and output lines are labeled with the exact differential values.

We will use the abbreviated format for differentials where a single character corresponds to an entire octet, and a 0 stands for a zero differential, and an X stands for a differential in the seventh bit of the octet. Hence, a differential of 00008000 (in hexadecimal) in a word would be abbreviated as 00X0. We will also list differentials only in the area that the attacker selects for the differential to occur. All other regions of the halfblocks always have zero differential.

Then, let us consider the following differential in the corresponding sections in the plaintext half-blocks, and where the spice has zero differential:

Left: X0X0 0000 X000 0000 0000 0000 00X0 0000 X0X0
Right: 0X00 X000 0000 0X00 0000 X000 0000 0X00 0X00

This differential goes through the pre-whitening without effect. Then, the right halfblock is presented to the F function. This differential is similar to the example differential given above, except that the single bit difference cycles through the 4 internal lines twice. This differential holds with probability 2^{-4} and produces this differential on the F output:

X000 0000 0000 0000 0000 0000 0000 0000 X000

Exclusive-oring that into the the left half-block, and swapping half-blocks gives as output of round 1 the following differential:

Left: 0X00 X000 0000 0X00 0000 X000 0000 0X00 0X00
Right: 00X0 0000 X000 0000 0000 0000 00X0 0000 00X0

The right halfblock is presented to the F function, and is somewhat more complex. This differential holds with probability 2^{-11} and produces this differential on the F output:

0X00 X000 0000 0X00 0000 X000 0000 0X00 0X00

It turns out that this differential is the same as the differential in the left half-block, and so, after exclusive-oring and swapping half-blocks, we get the following differential after two rounds:

Left: 00X0 0000 X000 0000 0000 0000 00X0 0000 00X0
Right: 0000 0000 0000 0000 0000 0000 0000 0000 0000

This is the trivial one-round differential, and so with probability 1, this produces a zero differential on the F output, and after swapping half-blocks, we get the following differential after round 3:

Left: 0000 0000 0000 0000 0000 0000 0000 0000 0000
Right: 00X0 0000 X000 0000 0000 0000 00X0 0000 00X0

The right halfblock differential is exactly the same as in round 2, and hence it, with probability 2^{-11} produces the same differential on the F output, and after the exclusive-or, and swapping half-blocks, we get the differential after round 4:

Left: 00X0 0000 X000 0000 0000 0000 00X0 0000 00X0
Right: 0X00 X000 0000 0X00 0000 X000 0000 0X00 0X00

As we will show below, a four round differential is sufficient for use as a distinguisher, and we have demonstrated one at total probability $2^{-(4+11+0+11)} = 2^{-26}$. However, if we do need another round, we can continue. The right halfblock differential is identical to the differential in round 1, and so with probability

2^{-4} produces the same differential on the F output, and after the exclusive-or, and swapping half-blocks, we get the differential after round 5:

Left: 0X00 X000 0000 0X00 0000 X000 0000 0X00 0X00
Right: X0X0 0000 X000 0000 0000 0000 00X0 0000 X0X0

This 5 round differential holds with a probability $2^{-(4+11+0+11+4)} = 2^{-30}$

Note that the characteristic formed from this differential are not formed by an iterated characteristic. Instead it is a specific concatenation of single round characteristics, where each characteristic leaves the cipher in exactly the state the next round characteristic requires.

4 Using This Differential

The most obvious use of this differential is as a distinguisher. The four round differential works as a distinguisher because the F function does not have complete diffusion. A 32 bit word at offset S of the F output is a function of the last 8 words and the words at offset 0 through S of the right halfblock, the spice and the key. With the above four round differential, at round five, all these inputs have zero differential for all offset S prior to the actual differential, as well as words 0 through S in the left halfblock. Therefor, the corresponding words in the left halfblock will still have zero differential after round 5, and since those words are never modified afterwards (other than xoring the postwhitening key) those words are output with zero differential. If we chose to put the input differential near the end of the halfblocks, the occurrence of the four round differential will cause the majority of one of the ciphertext halfblocks to have zero differential. This area is sufficiently large[7] that the probability of a zero differential of that size by coincidence is negligible.

We also note that this attack can be extended for variants of Mercy with more rounds. For example, the 5 round differential can be used in a boomerang configuration [2] to distinguish a 10 round Mercy variant from a random permutation with approximately 2^{122} chosen plaintexts and ciphertexts. In addition, there is a more complex differential over 6 rounds at probability 2^{-114}, which can be used to distinguish an 8 round Mercy variant from a random permutation using only chosen plaintexts. These show that, rather than adding additional rounds, modifying the F function would be recommended.

At first glance, this differential would appear to be sensitive to the precise organization of operation Q. To study this, a computerized search was done over a considerable number of variants of operation Q. Each variant studied had 1-4 operations where one of the internal lines were added or exclusive-ored into the leftmost data line, and where the the various operations were done in various orders. It turns out that all variants studied permitted differentials similar to the one presented in section 3. In addition, it turns out that the actual operation Q

[7] Possibly as large as 1504 bits

that Mercy used was optimal for variants with three operations in terms of differential probability, except for some variants that applied operation M directly to the data from the right halfside, and for some variants which used three additions, both of which possibly weaken the cipher for other reasons. When variants with an additional add or exclusive-or operation were considered, the optimal variants turned out to permit a four round differential with probability 2^{-52}, and one such variant shown on Figure 6 (and the other three are slight modifications of this). It is apparent that even this modified operation Q is still insufficient to prevent exploitable differentials.

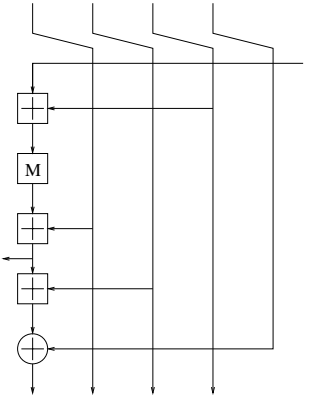


Fig. 6. Modified operation Q that yields a four round differential resistance of 2^{-52}.

5 Conclusions

We have shown a differential with a probability 2^{-26} that is detectable after 6 rounds. We have shown how to use this as a distinguisher which violates the Mercy design criteria.

This differential works because of two specific properties of the cipher. The first property is that, in spite of having a large number of sbox lookups during a block encryption, the actual number of sbox inputs are relatively small compared to the number of bits within the block itself. During a single block encryption, a total of 3264 bits are presented to the sbox. This would appear to be a large number, except there is a total of 4096 bits within the block, or in other words, there is less than a single sbox lookup per input octet. This attack took advantage of that relative paucity of sbox lookups by managing to control the sbox inputs while having nonzero differentials elsewhere in the cipher.

The other property that this attack used was the fact that, if we examine the effect of a differential within a sequential part of the right half-block, we find the effects can be described by the differential in the descending lines in the Q array

immediately afterwards and in the corresponding segment in the left half-block. If we managed to cancel out the differential immediately afterwards within the Q array, the only remaining effect was confined to an area that would, in the next round, modify the first area (and the descending outputs of that round's Q array). Hence, by repeatedly confining the differential, we were able to have a differential that was zero in most of the cipher, while two limited areas interacted with each other. If one were to modify the relationship between F inputs and outputs so that the effect of a differential could not be easily confined to two small areas, this should prevent such limited differentials. It would not, in fact, prevent large scale differentials, however, large scale differentials would have large numbers of additions with active differentials, and because each such addition[8] would stop the differential with nontrivial probability, a large number of them would make the differential hold with extremely low probability.

It will require more research to determine whether this attack can derive information on the internal d tables (or equivalently, the T function), or how to derive a Mercy variant that is immune to versions of this attack.

References

1. Crowley, P., "Mercy: a fast large block cipher for disk sector encryption", Proceedings of the Fast Software Encryption 2000, Springer-Verlag.
2. Wagner, D., "The boomerang attack", Proceedings of the Fast Software Encryption 1999, Springer-Verlag.

[8] Other than additions with differentials only in the MSBit.

Producing Collisions for PANAMA*

Vincent Rijmen**, Bart Van Rompay, Bart Preneel, and Joos Vandewalle

Katholieke Universiteit Leuven, ESAT-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{vincent.rijmen,bart.vanrompay}@esat.kuleuven.ac.be

Abstract. PANAMA is a cryptographic module that was presented at the FSE Workshop in '98 by Joan Daemen and Craig Clapp. It can serve both as a stream cipher and as a cryptographic hash function, with a hash result of 256 bits. PANAMA achieves high performance (for large amounts of data) because of its inherent parallelism. We will analyse the security of PANAMA when used as a hash function, and demonstrate an attack able to find collisions much faster than by birthday attack. The computational complexity of our current attack is 2^{82}; the required amount of memory is negligible.

1 Introduction

PANAMA is a cryptographic module that can be used in two modes, called *push* and *pull* mode. The module can be used both as a hash function and as a stream cipher. For a full description of PANAMA, we refer to [1].

In this paper we will describe a method to produce collisions for the PANAMA hash function. A collision occurs when two different messages are hashed to the same value. The PANAMA hash function maps messages of arbitrary length to a hash result of 256 bits, which means that a general birthday attack would need about 2^{128} operations fo find a colliding pair of messages. We will show that with our method a collision can be found with significantly less operations and a small amount of memory.

2 Short Description of PANAMA

The PANAMA stream/hash module has two types of internal memory: the *buffer* and the *state*. The buffer is a linear feedback shift register, containing 32 cells of 256 bits, denoted $b^0, \ldots b^{31}$. The state consists of 544 bits, divided in 17 32-bit words. Let the state of PANAMA be denoted by a 3-tuple (r, s, t), where r

* The work described in this paper has been supported in part by the Commission of the European Communities through the IST Programme under Contract IST-1999-12324 and by the Concerted Research Action (GOA) Mefisto-666.

** F.W.O. postdoctoral researcher, sponsored by the Fund for Scientific Research, Flanders — Belgium.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 37–51, 2002.

denotes the first 32-bit word of the state, and s and t each denote a 256-bit (8 words) value.

In the PANAMA hash mode, the input is padded to a multiple of 256 bits, and divided into blocks of 256 bits. All internal memory bits are set to zero, and then the following steps are executed for each message block. First, the state is updated by applying the nonlinear transformation ρ (which is composed of three specific transformations $\theta \circ \pi \circ \gamma$, described in the next section):

$$(r, s, t) \leftarrow \rho(r, s, t).$$

Secondly, the message block m and b^{16}, the contents of buffer cell 16, are exored into the state. The least significant bit of word 0 is flipped. These three operations are denoted here[1] by σ:

$$(r, s, t) \leftarrow \sigma(r, s, t) = (r + 1, s + m, t + b^{16}).$$

Thirdly, the message block is fed into the buffer and the LFSR is stepped once.

When all message blocks have been processed, 33 extra iterations are performed, but now the message input to the buffer is replaced by the state (part s), and the message input of σ is replaced by the contents of buffer cell 4. The 256-bit hash result is defined as the final state (part t).

Note that the buffer content is not present in the output. This means that there are two types of collisions for PANAMA: collisions in the state only, and collisions in both the buffer and the state. The 33 iterations after the last message block has been processed have as function to make it difficult to produce collisions of the first kind. We will try to find collisions of the second kind.

We will use the following notation: A message stream is denoted by $Pn, P(n-1), \ldots, P2, P1, P0$, where $P0$ is the first message block and Pn the last. Let X be an array of 8 32-bit words X_i, then $Y = r(X) \Leftrightarrow Y_i = X_{i+2 \bmod 8}$. Throughout this paper, '+' denotes bitwise addition (exor), except when used in the indices, where it denotes addition modulo 17 (if not explicitly stated otherwise).

3 Components of PANAMA

We describe here the different components of the nonlinear state updating transformation ρ. We also make some preliminary observations which will be useful later on. The transformations operate on the 17 words of the PANAMA state, where each word consists of 32 bits. In the following the index i ($0 \leq i < 17$) denotes the position of the word in the state.

3.1 The Transformation θ

The linear transformation θ is defined as follows:

$$c = \theta(a) \Leftrightarrow c_i = a_i + a_{i+1} + a_{i+4} \tag{1}$$

We calculated the inverse transformation θ^{-1}:

[1] Our notation differs with the one in [1], where σ is included in ρ.

$$a_i = c_{i+1} + c_{i+2} + c_{i+5} + c_{i+9} + c_{i+10} + c_{i+11} + c_{i+12} + c_{i+14} + c_{i+16}. \quad (2)$$

3.2 The Transformation π

The transformation π which combines cyclic word shifts with a permutation of the word positions is defined below. Here $[j]$ $(0 \le j < 32)$ denotes the bit position in a word, the multiplication $7i$ in the index is modulo 17 like the additions, and $r(i) = i \times (i+1)/2$ is a bit rotation which should be taken modulo 32.

$$c = \pi(a) \Leftrightarrow c_i[j] = a_{7i}[j + r(i)] \quad (3)$$

3.3 The Transformation γ

The transformation γ is the only nonlinear component. Because γ does not mix bits with different posititions in the words, it can be considered as a parallel application of 32 transformations γ_b.

Let a and $a + d$ denote two 17-bit vectors (containing one bit from each of the state words), that are input to γ_b. Let c and $c + e$ denote the corresponding outputs: $c = \gamma_b(a), c + e = \gamma_b(a + d)$. From the definition of γ, we have the following:

$$c_i = a_i + (a_{i+1} \vee (1 + a_{i+2})), \quad (4)$$

$$c_i + e_i = a_i + d_i + ((a_{i+1} + d_{i+1}) \vee (1 + a_{i+2} + d_{i+2})). \quad (5)$$

Using De Morgan's law, (4) can be transformed into

$$c_i = a_i + a_{i+1}a_{i+2} + a_{i+2} + 1. \quad (6)$$

Doing the same with (5) and combining the result with (6), we get

$$e_i + d_i + d_{i+2} + d_{i+1}d_{i+2} = a_{i+1}d_{i+2} + a_{i+2}d_{i+1}. \quad (7)$$

Table 1 below lists the solutions to this equation, in a format that is useful further on. It shows the relation between the absolute value of the input, the input difference and the output difference.

4 Message Format for Collisions

The linear updating function of the buffer imposes strict conditions on the format of colliding messages.

4.1 Collisions for the Buffer

The buffer has a linear updating function λ that is slightly more complex than in an ordinary LFSR, $b \leftarrow \lambda(b)$ is defined by:

Table 1. The transformation γ_b: relation between input difference (d), output difference (e) and absolute value of the input (a). An x or y in the column of a means that this bit can take two values, e.g. $(x, 1+x)$ means 'both $(0,1)$ and $(1,0)$ are possible.'

$d_i + e_i$	d_{i+1}	d_{i+2}	(a_{i+1}, a_{i+2})
0	0	0	(x, y)
0	0	1	$(1, x)$
0	1	0	$(x, 0)$
0	1	1	(x, x)
1	0	0	-
1	0	1	$(0, x)$
1	1	0	$(x, 1)$
1	1	1	$(x, 1+x)$

$$\begin{aligned} b^j &\leftarrow b^{j-1} \text{ if } j \notin \{0, 25\}, \\ b^0 &\leftarrow b^{31} + m, \\ b^{25} &\leftarrow b^{24} + r(b^{31}), \end{aligned}$$

where m is the message block that is fed into the buffer.

Therefore, the simplest collision for the buffer has the following difference pattern in the message stream (remember that the rightmost message block is the first):

$$dX, 24 \text{ zero cells}, r(dX), 0, 0, 0, 0, 0, 0, dX \ . \tag{8}$$

All difference patterns for buffer collisions are composed by adding shifted versions of pattern (8).

4.2 Collisions for the State

For a difference pattern of type (8), there will be five occasions where the input difference of the state is nonzero: the first two nonzero blocks are each injected twice in the state: once when they are the current message block and once when they pass through buffer cell 16. The last nonzero block cancels out all differences in the buffer, but is injected once in the state.

We are going to use a strategy of immediate compensation in the state: every time a difference is introduced in the state, we will try to let it die out as quickly as possible. A collision can then be seen as consisting of five 'sub-collisions', where a sub-collision is defined as a collision in the state only. This is also observed by the designers of PANAMA in [1].

Since the state update function ρ is invertible, a difference dX which is introduced can only disappear under the influence of another difference dY. An intuitive choice for the colliding messages format is the following:

$$dY, dX, 23 \text{ zero cells}, r(dY), r(dX), 0, 0, 0, 0, 0, dY, dX \ . \tag{9}$$

Since this format is the addition of two shifted copies of (8), it will produce a collision in the buffer. The strategy of immediate compensation demands that

the difference introduced in the state by the 'dX' values, is compensated by the 'dY' values. The difference propagation should be as follows:

$$\begin{array}{l}(0,0,0) \xrightarrow{\sigma} (0,dX,0) \xrightarrow{\rho} (0,dY,0) \xrightarrow{\sigma} (0,0,0)\\ (0,0,0) \xrightarrow{\sigma} (0,r(dX),0) \xrightarrow{\rho} (0,r(dY),0) \xrightarrow{\sigma} (0,0,0)\\ (0,0,0) \xrightarrow{\sigma} (0,0,dX) \xrightarrow{\rho} (0,0,dY) \xrightarrow{\sigma} (0,0,0)\\ (0,0,0) \xrightarrow{\sigma} (0,0,r(dX)) \xrightarrow{\rho} (0,0,r(dY)) \xrightarrow{\sigma} (0,0,0)\end{array} \tag{10}$$

Note that the first propagation path applies to the first and the fifth sub-collision. It turns out that due the interaction between the buffer update operation and the state update operation, there are no solutions dX, dY for (10). A proof is given in Appendix A. Therefore we will use a difference pattern of the following form:

$$dY, 0, dX, 22 \text{ zero cells}, r(dY), 0, r(dX), 0, 0, 0, 0, dY, 0, dX\ . \tag{11}$$

Table 2 gives a schematic overview of the difference values in the buffer and the state during the attack.

5 Overview of the Procedure

We will use differences where the bits of a 32-bit word are all set or all unset. This allows to denote a difference pattern in the state with 17 bits, one for each word. Furthermore, it means that the bit rotation in π has no effect on the difference values. We make this choice mainly because it is easier to think about differences of this format.

A collision will be found by combining the results of the searches for the five sub-collisions. The method we present here to find a sub-collision works for any value of the state. Note that in every sub-collision, the message input of σ is 'fresh'; it can be chosen freely. Since the message words are added to state words 1 to 8, it is easy to control that part of the state. State word 0 is exored with the constant 00000001_x so we have no direct control over its value. The buffer input of σ is added to state words 9 to 16. This input is influenced by the values of messages that have been injected in the state earlier on.

It seems difficult to make use of the fact that the buffer can be controlled: changing the buffer would require a recalculation of the current state. Our method assumes that only the message input can be controlled. Because the message input of σ influences only 8 of the 17 state words, we have not enough degrees of freedom if we vary the current message block only. Therefore, we will also have to select values for the common message block before message blocks with difference dX or $r(dX)$.

For every sub-collision, we use the following notation. The state will be denoted by the capitals A to N, the difference values by dA to dN.

$$\begin{array}{l}K \xrightarrow{\sigma 0} L \xrightarrow{\gamma} M \xrightarrow{\pi} N \xrightarrow{\theta} A \xrightarrow{\sigma 1} B \xrightarrow{\gamma} C \xrightarrow{\pi} D \xrightarrow{\theta} E\\ E \xrightarrow{\sigma 2} F \xrightarrow{\gamma} G \xrightarrow{\pi} H \xrightarrow{\theta} I \xrightarrow{\sigma 3} J\end{array} \tag{12}$$

Table 2. Schematic overview of the difference propagation through the buffer cells and the state. The shown values for the buffer and the state are the values *after* message block P_n has been processed. The nonzero differences in the state during subcollision i are denoted with z_i. Because of the strategy of immediate compensation, the differences in the state are zero most of the time.

n	P_n	Buffer										State		
		0	1	2	⋯	16	⋯	24	25	⋯	30	31	r	s	t
0	0	0	0	0		0		0	0		0	0	0	0	0
1	dX	dX	0	0		0		0	0		0	0	0	dX	0
2	0	0	dX	0		0		0	0		0	0	z_1	z_1	z_1
3	dY	dY	0	dX		0		0	0		0	0	0	0	0
4	0	0	dY	0		0		0	0		0	0	0	0	0
5	0	0	0	dY		0		0	0		0	0	0	0	0
6	0	0	0	0		0		0	0		0	0	0	0	0
7	0	0	0	0		0		0	0		0	0	0	0	0
8	$r(dX)$	$r(dX)$	0	0		0		0	0		0	0	0	$r(dX)$	0
9	0	0	$r(dX)$	0		0		0	0		0	0	z_2	z_2	z_2
10	$r(dY)$	$r(dY)$	0	$r(dX)$		0		0	0		0	0	0	0	0
11	0	0	$r(dY)$	0		0		0	0		0	0	0	0	0
12	0	0	0	$r(dY)$		0		0	0		0	0	0	0	0
13	0	0	0	0		0		0	0		0	0	0	0	0
14	0	0	0	0		0		0	0		0	0	0	0	0
15	0	0	0	0		0		0	0		0	0	0	0	0
16	0	0	0	0		0		0	0		0	0	0	0	0
17	0	0	0	0		dX		0	0		0	0	0	0	dX
18	0	0	0	0		0		0	0		0	0	z_3	z_3	z_3
19	0	0	0	0		dY		0	0		0	0	0	0	0
20	0	0	0	0		0		0	0		0	0	0	0	0
21	0	0	0	0		0		0	0		0	0	0	0	0
22	0	0	0	0		0		0	0		0	0	0	0	0
23	0	0	0	0		0		0	0		0	0	0	0	0
24	0	0	0	0		$r(dX)$		0	0		0	0	0	0	$r(dX)$
25	0	0	0	0		0		dX	0		0	0	z_4	z_4	z_4
26	0	0	0	0		$r(dY)$		0	dX		0	0	0	0	0
27	0	0	0	0		0		dY	0		0	0	0	0	0
28	0	0	0	0		0		0	dY		0	0	0	0	0
29	0	0	0	0		0		0	0		0	0	0	0	0
30	0	0	0	0		0		0	0		0	0	0	0	0
31	0	0	0	0		0		0	0		dX	0	0	0	0
32	0	0	0	0		0		$r(dX)$	0		0	dX	0	0	0
33	dX	0	0	0		0		0	0		dY	0	0	dX	0
34	0	0	0	0		0		$r(dY)$	0		0	dY	z_5	z_5	z_5
35	dY	0	0	0		0		0	0		0	0	0	0	0

We consider in total 4 σ operations, σn denoting the σ operation with message block Pn. Message blocks $P0$ and $P2$ are equal for both messages of the collision. For sub-collisions 1, 2 and 5 block $P1$ has the 'dX difference', and block $P3$ the

'dY' difference. For the other two sub-collisions these differences are imposed by the contents of buffer cell 16 in $\sigma 1$ and $\sigma 3$. The states K to A are equal in both messages, the common block $P0$ is used to bring the state to a value that allows a sub-collision to happen. It can be seen that the absolute value of the $P3$ blocks has no importance, for every sub-collision the difference in state I must be canceled by the difference in $P3$ (or in buffer cell 16 for $\sigma 3$).

6 The Chosen Difference Format

We give here a difference pattern that is a solution of (11). We describe the propagation of the difference from state B to I for every sub-collision. As noted above the pattern is decribed by 17 bits, one bit for each word ($0 \leq i < 17$).

The difference propagation through the linear transformations π and θ does not depend on the absolute value of the state, while $\sigma 2$ doesn't change the difference at all. The difference propagation through the nonlinear transformation γ depends on the absolute value of the state and this imposes a set of conditions on the absolute value of the state at 'time' B and F. These conditions, which can be derived using the results from Table 1, must be satisfied for the sub-collisions to occur. Note again that we can work with single bits because γ can be seen as 32 parallel transformations γ_b.

Table 3 below shows the required difference propagation for sub-collision 1 (and 5). In this table state B has the difference dX in part s, as imposed by message block $P1$. State I has the difference dY in part s, which will be canceled by message block $P3$. Parts r $(i = 0)$, s $(i = 1, \ldots 8)$ and t $(i = 9, \ldots 16)$ are clearly seperated in the table. Table 1 has been used to derive the following conditions on B and F for the difference propagation of Table 3.

$$B_5 = 1, B_6 + B_7 = 1, B_8 = 0, \tag{13}$$

$$\begin{aligned} &F_0 + F_1 = 1, F_2 = 1, F_3 = 0, F_5 = 0, F_6 = 0, F_7 = F_8, F_8 + F_9 = 1, \\ &F_{10} = 0, F_{11} = 0, F_{12} = F_{13} = F_{14}, F_{15} = 1, F_{16} = 1. \end{aligned} \tag{14}$$

For the other sub-collisions we refer to the similar Tables 4 to 5 and conditions (35) to (40) in Appendix B. All these sub-collisions are related because in every case state B contains the difference dX or $r(dX)$ in part s (imposed by the message) or part t (imposed by buffer cell 16), while state I must contain the difference dY respectively $r(dY)$ in the corresponding part of the state.

Other solutions for the difference format are possible. We have chosen one which results in the minimum number of conditions on B and F. It is not clear how to make an optimal choice for an easy solution of these conditions in the next step of the attack (section 7).

7 Producing Collisions

In order to generate each of the sub-collisions we need to solve the conditions for the absolute state values at 'time' B and F as given in section 6 and Appendix

Table 3. The required difference propagation for sub-collision 1 (and 5).

i	dB	dC	dD	$dE = dF$	dG	dH	dI
0	0	0	0	1	0	0	0
1	0	0	1	1	1	0	0
2	0	0	0	0	1	1	1
3	0	0	0	0	0	1	1
4	0	0	0	1	1	0	0
5	0	1	0	0	1	1	1
6	1	1	0	0	0	1	0
7	1	1	0	1	0	0	0
8	0	0	1	1	1	1	1
9	0	0	0	1	1	1	0
10	0	0	0	0	1	1	0
11	0	0	0	0	0	1	0
12	0	0	0	1	1	1	0
13	0	0	1	1	0	0	0
14	0	0	0	1	1	0	0
15	0	0	0	0	0	0	0
16	0	0	0	0	1	1	0

B. Values for state words 1 to 8 (part s) can be met by a suitable choice of the message block used at the corresponding time: $P1$ for B and $P2$ for F. To meet the values for the other state words (0 and 9 to 16, parts r and t) we need to calculate backwards to the previous message block: $P0$ for B and $P1$ for F. In this way the problem of finding a sub-collision can be solved in three steps:

1. solve a system of equations in the unknowns $L_1, \ldots L_8$, this determines message block $P0$,
2. solve a system of equations in the unknowns $B_1, \ldots B_8$, this determines message block $P1$,
3. solve a system of equations in the unknowns $F_1, \ldots F_8$, this determines message block $P2$.

The difficulty in solving these systems of equations is that when we calculate backwards to the previous message blocks, we obtain increasingly nonlinear equations because of the transformation γ. In our approach we only go back through one application of γ. To explain how we solve the problem, we will first describe the solution of the first sub-collision in some detail. Next we will summarize the procedure for the other sub-collisions and discuss the complexity of the attack.

7.1 Producing Sub-collision 1 (and 5)

From (14) it can be seen that there are 7 conditions in the unknowns $F_1, \ldots F_8$. Let $[j]$ denote the bit position, then we have the following equations for $0 \leq j < 32$:

$$\begin{array}{l} F_0[j] + F_1[j] = 1, F_2[j] = 1, F_3[j] = 0, F_5[j] = 0, \\ F_6[j] = 0, F_7[j] = F_8[j], F_8[j] + F_9[j] = 1. \end{array} \tag{15}$$

These conditions are easy to satisfy because $F_1, \ldots F_8$ are obtained by bitwise addition of $E_1, \ldots E_8$ and message block $P2$ which can be freely chosen.

In (14) we have 6 remaining conditions on $F_0, F_9, \ldots F_{16}$, which can be transformed into equations in the unknowns $B_1, \ldots B_8$. These have to be added to the 3 conditions we already have in these unknowns from (13),

$$B_5[j] = 1, B_6[j] + B_7[j] = 1, B_8[j] = 0, \tag{16}$$

resulting in an overdetermined system of 9 equations (and 32 bit positions). Generally such a system of equations has no solution, unless if one of the equations is dependent on the other eight. If there is no hidden structure in these equations, we can assume that we will need 2^{32} trials before we can solve this system of equations. Experimental evidence confirms this assumption.

To calculate backwards from F to B we apply the transformations σ, θ, π and γ, which lead to the following equations. Here $mb_i[j] = b^{16}_{i-9}[j]$ for $9 \leq i \leq 16$, $mb_0[j] = 0$ for $0 \leq j \leq 30$ and $mb_0[31] = 1$.

$$\begin{array}{l} F_i[j] = mb_i[j] + E_i[j] \quad (i = 0, 9 \ldots 16), \\ E_i[j] = D_i[j] + D_{i+1}[j] + D_{i+4}[j], \\ D_i[j] = C_{7i}[j + r(i)], \\ C_i[j] = B_i[j] + B_{i+1}[j]B_{i+2}[j] + B_{i+2}[j] + 1. \end{array} \tag{17}$$

Using these equations, for example the condition $F_{10}[j] = 0$ can be transformed in the following equation in the unknowns B_2, B_3 and B_4:

$$\begin{array}{l} \quad B_2[j+23] + B_3[j+23]B_4[j+23] + B_4[j+23] \\ + B_9[j+2] + B_{10}[j+2]B_{11}[j+2] + B_{11}[j+2] \\ + B_{13}[j+9] + B_{14}[j+9]B_{15}[j+9] + B_{15}[j+9] \\ + mb_{10}[j] + 1 = 0. \end{array} \tag{18}$$

It is because of the different bit rotations in this equation (and others), that we need to solve the equations bit per bit.

In order to simplify the system of 9 equations in $B_1, \ldots B_8$, we impose an extra condition $B_0[j] = 0$, which can be transformed into the equation:

$$\begin{array}{l} \quad L_0[j] + L_1[j]L_2[j] + L_2[j] \\ + L_7[j+1] + L_8[j+1]L_9[j+1] + L_9[j+1] \\ + L_{11}[j+10] + L_{12}[j+10]L_{13}[j+10] + L_{13}[j+10] \\ + mb_0[j] + 1 = 0, \end{array} \tag{19}$$

in the unknowns L_1, L_2, L_7 and L_8. Starting from an arbitrary but specified state K and buffer, the values $L_0, L_9, \ldots L_{16}$ are fixed, and condition (19) can easily be satisfied, by picking random values for $L_1, \ldots L_6, L_8$ and calculating L_7. This determines message block $P0$, and allows us to calculate states M, N, A and $B_0, B_9, \ldots B_{16}$.

Returning to the system of equations in $B_1, \ldots B_8$ we have already seen that there is a requirement on 32 bits. It turns out that there is another complication because by replacing $B_6[j]$ by $B_7[j] + 1$ (see (16)), we end up with an equation in two bit positions of the same state word B_7, of the form:

$$B_7[j + 27] + B_7[j + 1] = \ldots,$$

which translates to two requirements, one for bitwise addition of all values obtained with even j, and one for bitwise addition of all values obtained with odd j. This can be seen as a requirement on 2 bits. Hence the probability that a solution for this system of equations can be found is $1/2^{32+2}$. We can try random values of the initial state K until this happens and at that time message blocks $P0$ and $P1$ are determined. We can then calculate states C, D, E and $F_0, F_9, \ldots F_{16}$, and choose a suitable value of message block $P2$ to satisfy (15) (the conditions on $F_1, \ldots F_8$).

The complexity to find sub-collision 1 (or 5) with this procedure is 2^{34}.

7.2 Producing the Other Sub-collisions

Sub-collision 2. In this case there are 6 conditions in the unknowns $F_1, \ldots F_8$, which can easily be met by choosing message block $P2$. The 7 remaining conditions on $F_0, F_9, \ldots F_{16}$, are added to the 3 existing conditions in the unknowns $B_1, \ldots B_8$, resulting in an overdetermined system of 10 equations, which leads to a requirement for $2 \times 32 = 64$ bits. The system is simplified by specifying $B_0[j] = B_9[j] = 0$, which leads to 2 extra conditions in the unknowns $L_1, \ldots L_8$, which are easily solved with a suitable choice of message block $P0$. The nonlinearity of the system for $B_1, \ldots B_8$ imposes two more requirements to solve it: one for 2 bits (similar to the case for sub-collision 1), and another for 16 bits. This last requirement comes from an equation in the unknown B_8 of the following form:

$$B_7[j + 27]B_8[j + 27] + B_8[j + 27] = \ldots,$$

where B_7 has already been solved. It can be seen that for $B_7[j + 27] = 0$ we can calculate $B_8[j + 27]$, but for $B_7[j + 27] = 1$ all unknowns drop from the equation. On average this happens for 16 bits and we end up with a requirement that has to be satisfied in order to get a solvable system of equations. Hence the probability that a solution can be found is $1/2^{32+32+2+16}$, and the complexity for finding this sub-collision is 2^{82}.

Sub-collision 3. There are 7 conditions in the unknowns $F_1, \ldots F_8$, which can easily be met by choosing message block $P2$. The 7 remaining conditions on $F_0, F_9, \ldots F_{16}$, lead to 7 equations in the unknowns $B_1, \ldots B_8$. We simplify the system by specifying $B_0[j] = 0$, which added to the 3 existing conditions in $B_9, \ldots B_{16}$, leads to 4 equations in the unknowns $L_1, \ldots L_8$, which are easy to solve with a suitable choice of message block $P0$. The nonlinear system for $B_1, \ldots B_8$ can be solved with two requirements for 32 and 16 bits respectively.

The 32-bit requirement comes from the fact that one equation specifies the same unknown as a set of three other equations. The complexity for finding this sub-collision is $2^{32+16} = 2^{48}$.

Sub-collision 4. This case is similar to the previous one, except there are only 4 conditions in the unknowns $F_1, \ldots F_8$, and the system of 4 equations in the unknowns $L_1, \ldots L_8$ imposes another requirement of 32 bits, which raises the complexity to $2^{32+16+32} = 2^{80}$.

7.3 Complexity of the Attack

As seen above the complexities for finding the 5 sub-collisions are 2^{34}, 2^{82}, 2^{48}, 2^{80} and 2^{34} respectively. There is no problem in connecting the sub-collisions, as we only need an arbitrary initial state for each, which can be obtained by choosing random message blocks between the sub-collisions. So the total complexity for our collision-finding attack is determined by sub-collision 2 (which is the most difficult to find). This complexity is about 2^{82}.

We tested this attack for a reduced version of PANAMA where all words have a bitlength of 8 instead of 32 (so the hash has length 64), and this confirmed the given complexities (for the 8-bit version the complexity is about 2^{22}). Sub-collisions 1, 3 (and 5) were also tested for a 16-bit version which again confirmed the complexity.

We believe improvements to this attack are possible. First of all better methods to solve the systems of nonlinear equations can be looked for. Furthermore, our attack still has a lot of freedom. We can calculate further backwards which has the advantage that we get more unknowns, but the difficulty that we get more complex nonlinear equations.

8 Conclusion

We have presented a method for producing collisions for PANAMA. The complexity of our current attack is 2^{82}. Although this is still too high to actually find collisions for PANAMA, it is much faster than a general birthday attack which would require on the order of 2^{128} operations. Furthermore, we believe this attack can be improved, and because of the large degree of freedom we have, we think it is likely that with additional effort collisions can actually be found for PANAMA.

In order to improve the security of PANAMA, the design could be altered so that the message influences a smaller part of the state (e.g., exor only 4 message words into the state at every application of σ). Instead, more buffer cells could be used as input of σ. The consequence for the attacker is that he has a smaller degree of freedom, and therefore needs to go further backwards in the attack to obtain enough unknowns, which gives him more complex equations to solve. Preferrably, the different buffer cells that would be used as input of σ, should be selected in such a way that the different sub-collisions can no longer be treated

independently (cf. Table 2). Of course this would decrease the performance of the algorithm. Furthermore, there still is a fundamental problem that we have the freedom to choose message blocks, and are able to go backwards in the attack by choosing message blocks before the sub-collision. To obtain confidence in the security of the algorithm, it seems necessary to ensure that we need to consider so many message blocks for one sub-collision so that the attacks on the different sub-collisions interfere with each other, making this strategy impossible.

Acknowledgements. The authors wish to thank Lars R. Knudsen for helpful discussions.

References

1. J. Daemen and C.S.K. Clapp, "Fast hashing and stream encryption with PANAMA," *Fast Software Encryption, LNCS 1372,* S. Vaudenay, Ed., Springer-Verlag, 1998, pp. 60–74.

A Collisions in Two Steps Are Not Possible

We show here why there are no solutions for (9). In this case the 'timeline' (12) reduces to:

$$A \xrightarrow{\sigma 0} B \xrightarrow{\gamma} C \xrightarrow{\pi} D \xrightarrow{\theta} E \xrightarrow{\sigma 1} F \tag{20}$$

Here $\sigma 0$ introduces the 'dX difference', and $\sigma 1$ the 'dY difference' which must cancel the difference in state E for every sub-collision.

We start by considering the first sub-collision. We know that for any value of $P0$ and dX,

$$dB_i = 0, \text{ for } i = 0, 9, 10, 11, \ldots, 16. \tag{21}$$

Since γ does only mix nearby words it follows that

$$dC_i = 0, \text{ for } i = 9, 10, 11, \ldots 15. \tag{22}$$

Applying the definition of π gives the following:

$$dD_i = 0, \text{ for } i = 2, 4, 7, 9, 11, 14, 16. \tag{23}$$

With (2), this can be translated to the following conditions on dE_i:

$$\begin{aligned}
dE_3 + dE_4 + dE_7 + dE_{11} + dE_{12} + dE_{13} + dE_{14} + dE_{16} + dE_1 &= 0 \\
dE_5 + dE_6 + dE_9 + dE_{13} + dE_{14} + dE_{15} + dE_{16} + dE_1 + dE_3 &= 0 \\
dE_8 + dE_9 + dE_{12} + dE_{16} + dE_0 + dE_1 + dE_2 + dE_4 + dE_6 &= 0 \\
dE_{10} + dE_{11} + dE_{14} + dE_1 + dE_2 + dE_3 + dE_4 + dE_6 + dE_8 &= 0 \\
dE_{12} + dE_{13} + dE_{16} + dE_3 + dE_4 + dE_5 + dE_6 + dE_8 + dE_{10} &= 0 \\
dE_{15} + dE_{16} + dE_2 + dE_6 + dE_7 + dE_8 + dE_9 + dE_{11} + dE_{13} &= 0 \\
dE_0 + dE_1 + dE_4 + dE_8 + dE_9 + dE_{10} + dE_{11} + dE_{13} + dE_{15} &= 0
\end{aligned} \tag{24}$$

Fulfilling (9) also requires that $dF_i = 0, \forall i$. Without specifying $P1$ or dY_i, we have conditions on dE:

$$dE_i = 0, \text{ for } i = 0, 9, 10, 11, \ldots, 16. \tag{25}$$

Furthermore,

$$dE_i = dY_{i-1}, \text{ for } i = 1, 2, \ldots, 8. \tag{26}$$

Combining (24), (25) and (26), gives us the following conditions on dY:

$$\begin{aligned} dY_2 + dY_3 + dY_6 + dY_0 &= 0 \\ dY_4 + dY_5 + dY_0 + dY_2 &= 0 \\ dY_7 + dY_0 + dY_1 + dY_3 + dY_5 &= 0 \\ dY_0 + dY_1 + dY_2 + dY_3 + dY_5 + dY_7 &= 0 \\ dY_2 + dY_3 + dY_4 + dY_5 + dY_7 &= 0 \\ dY_1 + dY_5 + dY_6 + dY_7 &= 0 \\ dY_0 + dY_3 + dY_7 &= 0 \end{aligned} \tag{27}$$

This can be transformed into:

$$dY_2 = 0, dY_6 = dY_7, dY_5 = dY_1, dY_3 + dY_6 + dY_0 = 0, dY_4 + dY_1 + dY_0 = 0. \tag{28}$$

We can do the same excercise for sub-collision 3, where the same differences are injected in the state, but now via the buffer tap. We get the following:

$$dB_i = 0, \text{ for } i = 0, 1, 2, \ldots, 8 \tag{29}$$

$$dC_i = 0, \text{ for } i = 0, 1, 2, \ldots, 6 \tag{30}$$

$$dD_i = 0, \text{ for } i = 0, 3, 5, 8, 10, 13, 15. \tag{31}$$

$$\begin{aligned} dE_1 + dE_2 + dE_5 + dE_9 + dE_{10} + dE_{11} + dE_{12} + dE_{14} + dE_{16} &= 0 \\ dE_4 + dE_5 + dE_8 + dE_{12} + dE_{13} + dE_{14} + dE_{15} + dE_0 + dE_2 &= 0 \\ dE_6 + dE_7 + dE_{10} + dE_{14} + dE_{15} + dE_{16} + dE_0 + dE_2 + dE_4 &= 0 \\ dE_9 + dE_{10} + dE_{13} + dE_0 + dE_1 + dE_2 + dE_3 + dE_5 + dE_7 &= 0 \\ dE_{11} + dE_{12} + dE_{15} + dE_2 + dE_3 + dE_4 + dE_5 + dE_7 + dE_9 &= 0 \\ dE_{14} + dE_{15} + dE_1 + dE_5 + dE_6 + dE_7 + dE_8 + dE_{10} + dE_{12} &= 0 \\ dE_{16} + dE_0 + dE_3 + dE_7 + dE_8 + dE_9 + dE_{10} + dE_{12} + dE_{14} &= 0 \end{aligned} \tag{32}$$

Now we have that $dE_i = 0$, for $i = 0, \ldots, 8$ and $dE_i = dY_{i-9}$, for $i = 9, \ldots, 16$.

$$\begin{aligned} dY_0 + dY_1 + dY_2 + dY_3 + dY_5 + dY_7 &= 0 \\ dY_3 + dY_4 + dY_5 + dY_6 &= 0 \\ dY_1 + dY_5 + dY_6 + dY_7 &= 0 \\ dY_0 + dY_1 + dY_4 &= 0 \\ dY_2 + dY_3 + dY_6 + dY_0 &= 0 \\ dY_5 + dY_6 + dY_1 + dY_3 &= 0 \\ dY_7 + dY_0 + dY_1 + dY_3 + dY_5 &= 0 \end{aligned} \tag{33}$$

Combining these results with the results from sub-collision 1 (28), we get:

$$dY_0 = dY_1 = dY_2 = dY_4 = dY_5 = 0, dY_3 = dY_6 = dY_7. \tag{34}$$

Because sub-collison 2 and 4 will result in similar conditions on $r(dY)$, the only valid solution is $dY = 0$. It's a bit surprising that combining the results of sub-collision 1 and 3 alone does not suffice to reach this conclusion; we get 7 equations for each sub-collision, but they are not independent, e.g. 2 of the 7 equations from sub-collision 1 are redundant.

B Difference Propagation for Sub-collisions 2 to 4

We give here Tables 4 and 5 with the required difference propagation for sub-collisions 2, 3 and 4, as well as the corresponding conditions on the absolute value of the state at 'time' B and F. Sub-collision 1 was described in section 6, where we also discussed the relations between Tables 3 to 5.

Table 4. The required difference propagation for sub-collision 2.

dB	dC	dD	$dE = dF$	dG	dH	dI
0	0	0	0	1	1	0
0	0	0	0	1	0	1
0	0	0	1	1	0	0
0	1	1	1	0	1	1
1	1	0	1	1	1	0
1	1	0	0	0	1	0
0	0	0	0	1	1	1
0	0	0	1	0	1	0
0	0	1	1	1	0	1
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	1	1	1	0
0	0	0	0	0	1	0
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	0	0

The difference propagation in sub-collision 2, as specified in Table 4, leads to the following conditions for B:

$$B_3 = 1, B_4 + B_5 = 1, B_6 = 0, \tag{35}$$

and the following conditions for F:

$$\begin{aligned} &F_0 = 0, F_1 = 0, F_2 + F_3 = 1, F_3 = F_4, F_5 = 1, F_6 = 1, F_7 + F_8 = 1, \\ &F_9 = 1, F_{10} = 0, F_{12} = 0, F_{13} = 1, F_{14} = F_{15}, F_{15} + F_{16} = 1. \end{aligned} \tag{36}$$

Table 5. The required difference propagation for sub-collision 3 (left) and sub-collision 4 (right).

dB	dC	dD	$dE = dF$	dG	dH	dI
0	0	0	0	1	1	0
0	0	0	1	0	1	0
0	0	1	1	1	1	0
0	0	0	1	0	1	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	0	1	1	0	0
0	0	1	1	1	1	0
0	0	0	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	1
0	0	0	0	0	1	1
0	1	0	0	0	0	0
0	1	0	1	1	1	1
1	1	1	1	1	1	0
1	1	0	1	1	0	0
0	0	0	0	0	1	1

dB	dC	dD	$dE = dF$	dG	dH	dI
0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	0	1	0
0	0	0	0	1	1	0
0	0	0	0	1	1	0
0	0	0	1	1	0	0
0	0	0	0	1	0	0
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	1	1	0	1
0	1	0	1	0	0	0
0	0	0	0	1	1	1
1	1	0	1	0	1	0
1	1	0	1	1	1	0
0	0	1	1	1	1	1
0	0	0	1	0	1	0
0	0	1	1	1	0	1

The difference propagation in sub-collision 3, as specified in Table 5 (left), leads to the following conditions for B:

$$B_{13} = 0, B_{14} + B_{15} = 1, B_{16} = 0, \tag{37}$$

and the following conditions for F:

$$\begin{aligned} &F_0 = 1, F_1 + F_2 = 1, F_2 + F_3 = 1, F_4 = 0, F_5 + F_6 = 1, F_6 + F_7 = 1, F_7 = F_8, \\ &F_8 = F_9, F_9 + F_{10} = 1, F_{11} = 0, F_{12} = 1, F_{13} = F_{14} = F_{15}, F_{16} = 0. \end{aligned} \tag{38}$$

The difference propagation in sub-collision 4, as specified in Table 5 (right), leads to the following conditions for B:

$$B_{11} = 0, B_{12} = B_{13}, B_{14} = 0, \tag{39}$$

and the following conditions for F:

$$\begin{aligned} &F_0 = 1, F_4 = 0, F_6 = 1, F_7 = 0, F_8 + F_9 = 1, F_8 = F_{10}, \\ &F_{11} = 0, F_{12} + F_{13} = 1, F_{12} = F_{14}, F_{12} = F_{15}, F_{12} = F_{16}. \end{aligned} \tag{40}$$

The RIPEMD^L and RIPEMD^R Improved Variants of MD4 Are Not Collision Free

Christophe Debaert[1] and Henri Gilbert[2]

[1] DGA/DCE/CELAR
[2] France Télécom R&D

Abstract. In 1992, the cryptographic hash function RIPEMD, a European proposal, was introduced as an improved variant of the MD4 hash function. RIPEMD involves two parallel lines of modified versions of the MD4 compression function. Three years later, an attack against a reduced version of RIPEMD in which the first or the last round of the RIPEMD compression function is omitted was described by Hans Dobbertin, who also published in 1998 a cryptanalysis of MD4. In this paper, we present a method for finding collisions in each of the parallel lines of RIPEMD. The collision search procedure requires only a few seconds computing time. We show that although the modifications of the MD4 compression function Used in RIPEMD introduce additional constraints in the cryptanalysis as Compared with Dobbertin's attack of MD4, these modifications do not result in an increase of the collision search computation time. It is still an open question whether collisions can be found for the full RIPEMD function.

1 Introduction

A collision resistant hash function can be informally defined as an easy to compute but hard to invert function which maps a message of arbitrary length into a fixed length (m-bit) hash value, and satisfies the property that finding a collision, i.e. two messages with the same hash value, is computationally unfeasible. The best collision resistance one can hope to achieve with an m-bit function is bounded above by the $O(2^{m/2})$ complexity of a birthday attack. For most currently used hash functions, $m = 128$ or $m = 160$. Collision resistant hash functions represent a quite useful primitive in cryptography and are of frequent use for the purposes of message pre-processing in digital signature, commitment in public key authentication schemes, etc.

Most collision resistant hash function candidates proposed so far are based upon the iterated use of a so-called compression function, which maps a fixed length ($m + n$-bit) input value into a shorter fixed length m-bit output value. First padding data (which include filling bits and sometimes information such as the message length) is appended to the M message to be hashed as to obtain a padded message which length is a multiple of n, and then split into n-bit blocks M_1 to M_k. Denote the compression function by f and the hash function by h. The m-bit hash value $h(M)$ is computed using the recurrence: $H_0 = IV0$ (where

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 52–65, 2002.

IV0 is an m-bit constant initial value); for i=1 to k $H_i = f(H_{i-1}||M_i)$; $h(M) = H_k$. We are using in the sequel a terminology introduced by H. Dobbertin to distinguish two kinds of situations where two distinct (IV,M) and (IV',M') inputs have to the same image by the compression function of an iterated hash function: we will restrict the use of the term collision to the case where in addition IV=IV' and use the term pseudo collision otherwise. It was independently shown by Merkle and Damgard that if a compression function f is collision and pseudo collision resistant, then under some simple additional conditions on the padding rule, the associated hash function is collision resistant.

The most commonly used and (for the most recent ones) most trusted existing collision resistant hash functions do all belong to the MD-family of iterated hash functions, which includes MD4 (a 128-bit hash function proposed in 1990 by R. Rivest [9]), MD5 (a more conservative 128-bit hash function proposed in 1991 by R. Rivest [10]), RIPEMD (a 128-bit function which mixes the outputs of two parallel improved variants of MD4, proposed in 1995 by the European RIPE consortium [8]), RIPEMD-128 and RIPEMD-160 (128-bit and 160-bit variants of RIPEMD [6]), SHA (a 160-bit hash function proposed in 1992 by NIST), and SHA-1 (which was proposed by NIST as an improvement and replacement to the initial SHA [7] in 1994).

The hash functions of the MD family and the associated compression functions of MD family have been submitted to an extensive cryptanalytic investigation during the past years. Some collisions attacks were discovered on two of the three rounds of the MD4 compression function: an attack on the two last rounds by den Boer and Bosselaers [2] and an attack on the two first rounds leading to almost collisions on the full MD4, by Vaudenay [11]... This provided initial arguments in favor of moving from MD4 to MD5. Later on, Dobbertin established three major results on the cryptanalysis of the MD family of hash functions, namely (1) collisions for the full MD4 compression and hash functions that led to recommending the abandonment of its use (2) collisions for both the first and the last two rounds of the compression function of RIPEMD and (3) pseudo collisions on the whole MD5 compression [4], which do not lead to a collision of the associated MD5 hash function, but nevertheless invalidate the applicability of the Merkle-Damgard construction. Finally, Joux and Chabaud [1] discovered an attack allowing to find collisions for SHA in approximately 2^{61} SHA computations (instead of the about 2^{80} computations an ideal 160-bit hash function would require). As a consequence of these analyses, MD4 and SHA are no longer recommended, RIPEMD-128 seems to be a more conservative 128-bit hash function than MD5 and RIPEMD, and RIPEMD-160 and SHA-1 seem to be far from reach of known attack methods. In this paper we present a new result on the cryptanalysis of RIPEMD, which is to a certain extent complementary of the attack on the two-round version of RIPEMD described in [5]: we show how to construct collisions for the RIPEMD^L and RIPEMD^R three-rounds improved variants of MD4 used in the two parallel lines of computations of the RIPEMD compression function (in the left line and the right line respectively). The overall structure of our attack on RIPEMD^L and RIPEMD^R is close to the one in Dob-

bertin's attack on MD4 [3]. However, some of the MD4 modifications introduced in RIPEMD^L and RIPEMD^R prevent against too direct transpositions of the MD4 attack, and do substantially complicate the construction of collisions, so that the key steps of both attacks are quite different. The rest of this paper is organized as follows. Section 2 briefly describes the RIPEMD, RIPEMD^L and RIPEMD^R compression functions (a full description of which is provided in appendix), and introduces some basic techniques applicable to the cryptanalysis of hash functions of the MD family. Section 3 provides an overview of our attack strategy and of the main steps of the attack, which are later on detailed in Sections 5, 6 and 7. Section 8 concludes the paper.

2 Preliminaries

2.1 Notation

Throughout this paper, all the operations or functions considered relate to 32-bit words. The + symbol represents a modulo 2^{32} addition, the $\oplus$, $\wedge$, and $\vee$ symbols represent the bitwise exclusive OR, bitwise AND and bitwise OR respectively. If X is a 32-bit word and $s \in [0..31]$ then $X^{<<s}$ denotes the left cyclic shift of X by s bit positions to the left.

2.2 Description of the RIPEMD, RIPEMD^L, and RIPEMD^R Compression Functions

The RIPEMD compression function transforms a 4-word (128-bit) initial state value $IV = (IV_A, IV_B, IV_C, IV_D)$ and a 16-word message block $X = (X_0, .., X_{15})$ into a 128-bit output value (AA, BB, CC, DD). As said before, it consists of two parallel lines of computation achieving two variants of the MD4 compression function, namely the RIPEMD^L and RIPEMD^R three round compression functions.

The RIPEMD^L (resp RIPEMD^R) register values (A, B, C, D) (resp $(A'B'C'D')$) are initially set to the IV value. Each of the three rounds of RIPEMD^L (resp RIPEMD^R) consists of 16 steps. Each step consists of a transformation of the form

$$A = (A + f_r(B, C, D) + X_{\varphi(i \bmod 16)} + K_r)^{<<s_i}$$

$$A' = (A' + f_r(B', C', D') + X_{\varphi(i \bmod 16)} + K'_r)^{<<s_i}$$

where $i \in [0..47]$ denotes the step number, $r \in [0..2]$ denotes the round number, f_r is a round dependent 32-bit bitwise boolean function of three inputs (namely the majority function in the first round, the multiplexing function in the second round and the exclusive or in the third round; φ is a round-dependent permutation of i mod 16, (namely the identity permutation of in the first round and two other permutations denoted by σ and ρ in the two other rounds), so that each message word is involved in exactly one step of each round; s_i is a rotation amount which depends upon the step number and K_r and K'_r are round

dependent constants: they represent the single element which causes RIPEMD^L and RIPEMD^R to differ. Finally the RIPEMD (resp RIPEMD^L and RIPEMD^R) outputs are deduced from the (A, B, C, D) and (A', B', C', D') states obtained at the completion of step 47.

A detailed description of the above compression functions is provided in appendix.

The problem of finding an (X, X^*) pair of colliding messages for a compression function of the MD family such as say RIPEMD^L can be restated in terms of controlling the $(\Delta A, \Delta B, \Delta C, \Delta D) = (A, B, C, D) - (A^*, B^*, C^*, D^*)$ differences between the register values induced by X and X^* at the various steps of the computation.

3 Outline of Our Attack

In this Section we provide an outline of the overall structure of our attack on the RIPEMD^L and RIPEMD^R compression functions. Our aim is to construct a collision of the form $f(IV0, X) = f(IVO, X^*)$ where $IV0$ is defined as the proposed initial value for RIPEMD [8] and $X = (X_i), i = 0..15$ and $X^* = X_i^*, i = 0..15$ are two 16-word messages. We further require that only one X word X_{i_0} be distinct from the corresponding X^* word $X_{i_0}^*$, and that the $\Delta X_{i_0} = X_{i_0} - X_{i_0}^*$ difference be of weight 1 (in other words there exists an integer $k \in [0..31]$, such that $\Delta X_{i_0} = 1^{<<k}$).

3.1 Attack Structure

There are three occurrences of the X_{i_0} (resp $X_{i_0}^*$) word in the f computation, namely at steps i_0 , $16 + \rho(i_0 \bmod 16)$, $32 + \sigma(i_0 \bmod 16)$, which respectively belong to the first, the second and the third round[1] of f [table 1].

Due to the fact that each step represents a one to one transformation of the (A, B, C, D) register, the collision must necessarily happen at occurrence 3 of X_{i_0}.

Our overall attack strategy is similar (at least at the level of detail considered in this Section) to the one used in Dobbertin's MD4 attack [3]. It consists in trying to enforce a suitably selected "almost collision", i.e. a suitably chosen low weight difference value, just after occurrence 2 of the X_{i_0} (resp $X_{i_0}^*$) message words, in such a way that with sufficiently high probability this almost collision results, just before occurrence 3, in an intermediate difference value which is compensated during that step by the ΔX_{i_0} message difference. The collision is then preserved between both computations after occurrence 3.

A little bit more in detail, our attack consists (once a suitable choice of i_0 has been found) of three main parts, which purpose is to efficiently generate allocations of the 16 X_i words in such a way that the probability of obtaining a full collision for one of these allocations is high.

[1] ρ and σ are defined at the end of the article.

Table 1. The three main parts of the attack.

IV0
Part 3: (backward adjustment)
Occurrence 1 of $X_{i_0}, X^*_{i_0}$ (step $i_0 = 10$)
Part 2: (inner almost-collisions search)
Occurrence 2 of $X_{i_0}, X^*_{i_0}$ (step $16 + \rho(i_0 \bmod 16) = 20$)
Part 1: (differential part)
Occurrence 3 of $X_{i_0}, X^*_{i_0}$ (step $32 + \sigma(i_0 \bmod 16) = 33$)

(1) Differential part [occurrences 2 to 3]

This preliminary part of the attack consists in finding a suitable Δ low weight intermediate difference value such that if $(\Delta A, \Delta B, \Delta C, \Delta D) = \Delta$ after occurrence 2 of the ΔX_{i_0} message difference, then with a sufficiently high probability $(\Delta A, \Delta B, \Delta C, \Delta D) = 0$ after occurrence 3. It is based upon quite simple differential cryptanalysis techniques.

(2) Almost collision search [occurrences 1 to 2]

This is the most complex part of the attack. It consists of finding an allocation of the $A_{i_0}, B_{i_0}, C_{i_0}, D_{i_0}$ registers value at the end of step i_0 and allocations of the X_i words corresponding to those i values involved in steps i_0 (occurrence 1) to $16 + \rho(i_0 \bmod 16)$ (occurrence 2) which together ensure that an almost collision occurs at occurrence 2, i.e. that the difference value on the registers A to D is equal to the Δ low weight difference found in part (1).

(3) Backward adjustment [beginning to occurrence 1]

This part of the attack consists of finding allocations of the X_i values not fixed in part (2) such that initial value IV0 results in the $A_{i_0}, B_{i_0}, C_{i_0}, D_{i_0}$ register values at the completion of occurrence 1 (step i_0).

The actual collision search procedure consists of first achieving the precomputations of parts (1) and parts (2) above, which determine a first subset of the X_i allocations, and then of using part (3) to efficiently generate many allocations of the other X_i values leading to an almost collision after occurrence 2, until eventually one of these almost collisions provides a full collision after occurrence 3.

3.2 Selecting an Appropriate i_0 Value: Reasons for the $i_0 = 10$ Choice

The selected i_0 value must achieve the best possible trade-off between the requirements inherent to the different parts of the attack, which can be summarized as follows:

- Part 1 (differential part): the number of steps between occurrences 2 and 3 of X_{i_0} must not be too large in order for the probability of the differential characteristic used in part 1 to be large enough. More importantly, the $\sigma(i_0 \bmod)$ number of steps where the H function (exclusive OR) is used must be extremely low, because otherwise the very fast diffusion achieved by H would result in unacceptably small collision probabilities.

- Part 2 (almost collisions search): the $16+\rho(i_0 \bmod 16)-i_0$ number of steps between occurrences 1 and 2 of X_{i_0} must be large enough (say at least 6), but not too large in order to leave some free variables for part 3 of the attack, and if possible not involve a not too large fraction of F functions (multiplexing), which are harder to exploit than the G function (majority).

- Part 3 (backward adjustment): as said before, the number of X_i variables left free after part 2 must be sufficient. Due to the modifications introduced in the RIPEMD ρ and σ permutations as compared with those involved MD4, it is impossible to find any i_0 value which is as suitable for an attack as the $i_0 = 12$ value used in the attack of MD4.

The part 1 requirement on a low $\sigma(i_0 \bmod 16)$ number of steps involving the H functions between occurrences 2 and 3 is very demanding, and leads to discarding candidate index value such as $i_0 = 15$. The $i_0 = 13$ index value had also to be discarded because the number of steps between occurrences 1 and 2 of X_{i_0} is only 5, a too low value. The index value $i_0 = 10$ appeared to realize the best trade-off between the various requirements: the number of steps between the end of occurrence 2 (step 21) and the end of occurrence 3 (step 34) is 13, a rather low value (note that the corresponding number of steps in the MD4 attack is 15), and more importantly only 3 of these 13 steps involve the H function; finally, the number of steps between occurrence 1 (step 10) and occurrence 2 (step 20) is acceptable (11 steps, which involve the 9 X_i input words 1, 4, 7, and 10 to 15).

The difference between X_{10} and X^*_{10} is chosen equal to the low Hamming weight value $1^{<<6}$. The attack aims at finding a collision for two collections of words X and X^* defined by setting:

$$X^*_i = X_i \text{ for } i \neq 10,\ X^*_{10} = X_{10} + 1^{<<6}$$

Each of the 3 parts ends with an occurrence of X_{10}. The first occurrence of X_{10} introduces differences in the computation of the collision. This differences in the registers are compensated by the difference of the third occurrence of X_{10} at step 33 and are equal to zero. So between the first and the second occurrences of the word 10, the differences have to be controlled to allow the compensation of the variations between steps 21 and 33.

To achieve this, a well-chosen value for the differences at step 20, $\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6})$, is required.

4 Differential Attack Modulo 2^{32} (Steps 21-33).

The Δ_{20} differences value just before step 21 must result after step 32 in an intermediate difference value compensated at step 33 by the $\Delta X_{10} = 1^{<<6}$ difference. This part of the attack is a routine differential attack.

The variations on the register occurring at any computation step i come from the difference on the inputs X_i and X_i^*, the direct diffusion of the changing register and the indirect diffusion from the boolean vector function. The difference on the inputs occurs only when X_{10} is used in the step (steps 10, 20 and 33). The direct diffusion cannot be suppressed whereas the differences due to indirect diffusions have a non zero probability to appear for the multiplexing (F) and Majority (G) functions. Thus, with a certain probability, the indirect diffusions are equal to zero in those steps belonging to the second round of the compression function (21-31) and compensate the variations during the third round steps (31-33).

The probabilities can be computed backward from step 33 to step 21. We obtain at the beginning of step 21, a $\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6})$ difference leading to no difference at the end of step 33, $\Delta_{33} = (0, 0, 0, 0)$. Table 2 gives for step i, the probability for Δ_i to be equal to the values in the table under the assumption that Δ_{i-1} does the same.

Table 2. Differential attack.

step	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
A				$1^{<<24}$				$1^{<<31}$				$1^{<<6}$				0	
B			0				0				0				0		
C		0				0				0				0			
D	$-1^{<<6}$				$-1^{<<15}$				$-1^{<<27}$				$-1^{<<6}$				0
P	1	1	1	1	1	1/9	1/9	1/3	1/3	1/9	1/9	1/3	1/3	1/9	1/9	1/3	1

The probabilities in table 2 are an approximation calculated without taking the correlations between the lines into account. Each step is strongly dependent on the 3 previous steps, but an exact computation of the probability would require lengthy considerations.

The differential attack cannot be achieved unless an initial constraint is satisfied. Actually, in an inner almost-collision, the registers are given fixed values. So registers from 18 to 20 are known and their values fix the probability of a computation with no indirect diffusion at step 21, to 0 or 1. A constraint is added. We reflect this additional condition in saying that in order for an inner almost-collision to be admissible, the following equations must be satisfied:

$$G(A_{20}, B_{19}, C_{18}) = G(A_{20}^*, B_{19}^*, C_{18}^*) \quad (1)$$

$$\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6}) \quad (2)$$

Suppose we have an admissible inner-collision. Then the probability[2] to achieve the differential attack is smaller than but close to $2^{-26.95}$.

5 Backward Adjustment (Steps 0 to 9)

Suppose we have found an admissible inner almost-collision. The method used to find an inner almost-collision corresponds to the resolution of a system where the parameters are the 32-bit registers used at steps 8 to 20 and the variation of the input word X_{10}. Solving the system fixes the contents of the registers R_8 to R_{20}. So using the equations of steps 12 to 20, we obtain the values of words 1, 4, 7, 10, 12, 13, 14 and 15.

The input word X_{11} appears in step 11. But a solution of the almost collision does not fix B_7 which X_{11} depends upon. So it does not fix X_{11}. Nevertheless, since the word X_{10} is fixed, B_7 and C_6 are largely linked and X_{11} cannot be considered as a free variable:

$$C_6 = C_{10}^{<<18} - X_{10} - F(D_9, A_8, B_7) - K_0 \tag{3}$$

There are still 7 free variables left for the rest of the attack. This is 2 variables less than in the cryptanalysis of MD4. But since the differential part of the attack requires numerous trials, it must be possible to take arbitrary values for some of the input words. For this, we randomly select X_0, X_2 and X_3. Since X_1 and X_4 are fixed by the admissible inner almost-collision found before, we can compute steps 0 to 4 to obtain the register values 0 to 4. Thus the backward adjustment problem consists in finding the remaining free variables, namely D_5, X_5, X_6, X_8 and X_9.

Let's fix $C_6 = -1$, set B_7 and compute X_{11}. The equations of step 7 and 10 become:

$$D_5 = B_7^{<<23} - X_7 - B_3 - K_0 \tag{4}$$

$$F(D_9, A_8, B_7) = C_{10}^{<<18} - X_{10} - C_6 - K_0 \tag{5}$$

Due to the involvement of the F boolean function (multiplexing) in the above equation , B_7 is not easy to compute after the inner almost-collision search. So, the equation (5) is handled as a new constraint for the resolution of the admissible inner almost-collision search system. It must be satisfied before the beginning of the computation for the right initial value.

Finally we know D_5 from equation (4) above, i.e. we know all registers from 0 to 20. At steps 5, 6, 8 and 9, the values of X_5, X_6, X_8 and X_9 compensate the contents of the registers.

This means we can reach the connection to a given inner almost-collision from a given initial value and keep 3 words free for the differential attack if the given inner almost-collision satisfies (5).

[2] This probability has to be compared with $2^{-30.11}$, the probability found in the cryptanalysis of MD4.

Explicitly, the prescribed initial value is matched by the settings:

$$A_4 = (A_0 + X_4 - F(B_3, C_2, D_1) + K_0)^{<<5} \tag{6}$$
$$C_6 = 0xffffffff \tag{7}$$
$$D_5 = B_7^{<<23} - B_3 - X_7 - K_0 \tag{8}$$
$$X_5 = D_5^{<<24} - D_1 - F(A_4, B_3, C_2) - K_0 \tag{9}$$
$$X_6 = C_6^{<<25} - C_2 - F(D_5, A_4, B_3) - K_0 \tag{10}$$
$$X_8 = A_8^{<<21} - A_4 - F(B_7, C_6, D_5) - K_0 \tag{11}$$
$$X_9 = D_9^{<<19} - D_5 - F(A_8, B_7, C_6) - K_0 \tag{12}$$

6 Inner Almost-Collisions

6.1 Steps 10 to 20: A System with Constraints

In this Section, we consider the compression function between the first two occurrences of X_{10} and X_{10}^* (10-20) and search an inner almost-collision between steps 10 and 20. The inner almost-collision has to be admissible and to satisfy the constraints resulting from the backward adjustment and differential parts of the attack:

$$C_6 = -1 \tag{13}$$
$$F(D_9, A_8, B_7) = C_{10}^{<<18} - X_{10} + 1 - K_0 \tag{14}$$
$$G(A_{20}, B_{19}, C_{18}) = G(A_{20}^*, B_{19}^*, C_{18}^*) \tag{15}$$
$$\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6}) \tag{16}$$

Thus, finding an inner almost-collision can be expressed as solving a system of equations under these four Constraints[3]. For each step from 10 to 20, an equation involving only the ΔX_j difference values, not the X_j values themselves, is provided by eliminating the value of the words.

$$X_j = a_{j+1}^{<<32-s_j} - (a_j + f_i(b_j, c_j, d_j) + K_j) \tag{17}$$
$$X_j^* = a_{j+1}^{*<<32-s_j} - (a_j^* + f_i(b_j^*, c_j^*, d_j^*) + K_j) \tag{18}$$
$$\Longrightarrow \Delta X_j = \Delta a_{j+1}^{<<32-s_j} - \Delta a_j - \Delta f_i(b_j, c_j, d_j) \tag{19}$$

Adding the equation (15) and using equation (16), we obtain a system of 12 (19)-like equations.

The resolution of this system fixes the contents of the registers from 8 to 20. Once the register values are known, the values of the words 10 to 15, 1, 4 and 7 can be deduced, using the (17)-like equations.

As the words X_{10} and X_{13} appear twice between steps 10 and 20, the X_{10} (resp X_{13}) values provided by the equations of steps 10 and 20 (resp 13 and 18)

[3] there were only two constraints in MD4 cryptanalysis.

must be equal. Thus two constraints must be added to the system in order for the register values to provide consistent X_{10} and X_{13} values. be added to

Therefore, an admissible inner almost-collision can be found by solving the system under the additional two constraints:

$$A_{20}^{<<21} - (A_{16} + G(B_{19}, C_{18}, D_{17}) + K_{20}) = C_{10}^{<<18} - (G(D_9, A_8, B_7) + K_{10} - 1)$$
$$D_{13}^{<<25} - (D_9 + F(A_{12}, B_{11}, C_{10}) + K_{13}) = C_{18}^{<<24} - (C_{14} + G(D_{17}, A_{16}, B_{15}) + K_{18})$$

6.2 Resolution

The resolution of the system and constraints defined above aims at finding the suitable values for the contents of the registers from B_7 to A_{20} and differences on registers R_{11} to R_{20}. The registers after 21 are computed by choosing randomly X_0, X_2 and X_3, whereas the registers before 6 are fixed by the backward adjustment as described previously.

We have to control the differences in the registers between the two first occurrences of the word 10, from $\Delta_9 = (0, 0, 0, 0)$ to $\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6})$. There are two ways to make the choices on the registers. On the one hand, the content of a register can be fixed to simplify the system, this specialization of the system reduces strongly the choices on the content of the register. On the other hand, at first, only the differences in the registers are chosen as to ensure consistency in the diffusions and afterwards the registers are given values. This leads to more possible values for a register.

In the resolution of the system, after suitable "specializations", solutions can be found by iterative solving of equations of the form:

$$G(Z + dZ, a_1, b_1) - G(Z, a_2, b_2) = C$$

where the Z unknown is a 32-bit word, G is a derived from a ternary boolean function and dZ, $a1$, $b1$, $a2$, $b2$ and C are known words. Equations are solved recursively and bitwise from the lowest bit to the highest.

Let's continue the backward computation of the differences begun in the differential part of the attack. We obtain the differences for registers from R_{13} to R_{20} [table 3] and simplify the system.

Table 3. Diffusion.

step		13	14	15	16	17	18	19	20
A	specialization				$1^{<<13}$				$1^{<<24}$
B	and			0				0	0
C	resolution		0				0		0
D		-1				$-1^{<<6}$			$-1^{<<6}$

Since the first equations of the system involve the multiplexing function F, the indirect diffusions are far more difficult to control. We specialize this part of the system by choosing $D_{13} = -1$ and $D_{13}^* = 0$.

Without the two additional constraints, the system can be rewritten:

$$D_{13} = -1 \ , \ D_{13}^{*} = 0 \tag{20}$$
$$\Delta C_{10}^{<<18} = 1^{<<6} \tag{21}$$
$$\Delta F(C_{10}, D_9, A_8) = \Delta B_{11}^{<<17} \tag{22}$$
$$\Delta F(B_{11}, C_{10}, D_9) = \Delta A_{12}^{<<26} \tag{23}$$
$$\Delta F(A_{12}, B_{11}, C_{10}) = -1 \tag{24}$$
$$A_{12}^{*} = B_{11} + \Delta C_{10} \tag{25}$$
$$\Delta F(C_{14}, D_{13}, A_{12}) = -\Delta B_{11} \tag{26}$$
$$B_{15} = C_{14} \oplus (1^{<<31} - \Delta A_{12}) \tag{27}$$
$$\Delta G(A_{16}, B_{15}, C_{14}) = 0 \tag{28}$$
$$\Delta G(D_{17}, A_{16}, B_{15}) = 0 \tag{29}$$
$$\Delta G(C_{18}, D_{17}, A_{16}) = 0 \tag{30}$$
$$\Delta G(B_{19}, C_{18}, D_{17}) = -1^{<<6} \tag{31}$$
$$\Delta G(A_{20}, B_{19}, C_{18}) = 0 \tag{32}$$

The end of the system (from equation (28)) can be solved as it is done in MD4 cryptanalysis: except in equation 31 (step 20), the diffusions are direct as it is in the differential attack and at step 20, the diffusion from the input are compensated by the indirect diffusion[4], whereas the direct diffusion introduces the variations on registers R_{20} required for the rest of the complete attack.

Only the variations for the registers R_{10}, R_{11} and R_{12} are still free. Nevertheless, we know $\Delta C_{10}^{<<18} = 1^{<<6}$. By choosing C_{10} and C_{10}^{*} to maximize the Hamming weight of the ΔC_{10} difference, we obtain $\Delta C_{10} = 1^{<<20} - 1^{<<14} + 1$ and maximize the possibilities of solving the rest of the system.

7 Collision Search Algorithm

We can now describe the collision search algorithm for RIPEMD^L.

There is a practical algorithm which requires less than 1 second and allows us to compute an inner almost-collision that satisfies the four equations:

$$C_6 = -1$$
$$F(D_9, A_8, B_7) = C_{10}^{<<18} - X_{10} + 1 - K_{10}$$
$$\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6})$$
$$\Delta G(A_{20}, B_{19}, C_{18}) = 0$$

[4] We can notice that since registers R_{17} has the same variation as words 10, the majority boolean function leads to a high probability to have a solution for register R_{19}.

1. Choose C_{10} and C^*_{10} that satisfy $\Delta C_{10} = 1^{<<20} - 1^{<<14} + 1$.
2. Choose A_{12}, ΔA_{12} and ΔB_{11}. Verify $\Delta F(A_{12}, B_{11}, C_{10}) = -1$.
3. Compute registers from 9 to 18 and verify the first constraint to obtain the value of X_{13}.
4. Compute registers R_7, R_8, R_{19} and R_{20}. Then verify the second constraint to obtain the value of X_{10}.
5. With the contents of the registers, set the value for X_{11}, X_{12}, X_{14}, X_{15}, X_1, X_4 and X_7. We obtain an admissible inner almost-collision.
6. Choose randomly X_0, X_2 and X_3.
7. Compute the registers for 0 to 5 as described in section 5. We reach the right initial value.
8. Achieve the differential part of the attack by making new trial for X_0, X_2 and X_3 until we reach a collision $\Delta_{33} = (0, 0, 0, 0)$.

With RIPEMD^L, we obtained a collision for the (X, X^*) pair determined by the following values:

$X_0 = 0x10978d07$	$X^*_0 = 0x10978d07$	$X_8 = 0xd0f6edc3$	$X^*_8 = 0xd0f6edc3$
$X_1 = 0x937e0e67$	$X^*_1 = 0x937e0e67$	$X_9 = 0xd22c0042$	$X^*_9 = 0xd22c0042$
$X_2 = 0x697a5fe9$	$X^*_2 = 0x697a5fe9$	$X_{10} = 0x847dc027$	$X^*_{10} = 0x847dc067$
$X_3 = 0x313d55b0$	$X^*_3 = 0x313d55b0$	$X_{11} = 0x84fbc027$	$X^*_{11} = 0x84fbc027$
$X_4 = 0xa816066b$	$X^*_4 = 0xa816066b$	$X_{12} = 0xfc7f4140$	$X^*_{12} = 0xfc7f4140$
$X_5 = 0xed3f60d6$	$X^*_5 = 0xed3f60d6$	$X_{13} = 0xfff00001$	$X^*_{13} = 0xfff00001$
$X_6 = 0xc6c1e1a$	$X^*_6 = 0xc6c1e1a$	$X_{14} = 0x7fff82$	$X^*_{14} = 0x7fff82$
$X_7 = 0xa58dc669$	$X^*_7 = 0xa58dc669$	$X_{15} = 0xffbf43$	$X^*_{15} = 0xffbf43$

With RIPEMD^R, we obtained a collision for the (X, X^*) pair determined by the following values:

$X_0 = 0x3e00cc54$	$X^*_0 = 0x3e00cc54$	$X_8 = 0xa32b5a9c$	$X^*_8 = 0xa32b5a9c$
$X_1 = 0xc6008000$	$X^*_1 = 0xc6008000$	$X_9 = 0xd6e17c84$	$X^*_9 = 0xd6e17c84$
$X_2 = 0x4f269ad0$	$X^*_2 = 0x4f269ad0$	$X_{10} = 0xc0003440$	$X^*_{10} = 0xc0003480$
$X_3 = 0x398358a6$	$X^*_3 = 0x398358a6$	$X_{11} = 0xc07e3440$	$X^*_{11} = 0xc07e3440$
$X_4 = 0x1448002$	$X^*_4 = 0x1448002$	$X_{12} = 0xabdcb55a$	$X^*_{12} = 0xabdcb55a$
$X_5 = 0xcc41f9d9$	$X^*_5 = 0xcc41f9d9$	$X_{13} = 0xaf4d741b$	$X^*_{13} = 0xaf4d741b$
$X_6 = 0xb0304fed$	$X^*_6 = 0xb0304fed$	$X_{14} = 0xafdd739c$	$X^*_{14} = 0xafdd739c$
$X_7 = 0x104002$	$X^*_7 = 0x104002$	$X_{15} = 0xb05d335d$	$X^*_{15} = 0xb05d335d$

8 Conclusion

It turns out that the modifications of the MD4 compression function introduced in each of the RIPEMD^L and RIPEMD^R compression functions of RIPEMD leads to more constraints in the cryptanalysis as compared with the Dobbertin's cryptanalysis attack of MD4, but collisions can still be found easily (smaller than $2^{-26.95}$ for the differential part of the attack). Our attack provides some arguments in favor of the conjecture that the existence of an attack on MD4 is

not due to a suboptimal selection of parameters such as the σ and ρ, the rotation amounts, etc. The selection made in RIPEMD does not lead to a stronger version of MD4.

Collisions could be found for each line of the RIPEMD compression function and for the two-round versions of RIPEMD described in [5]. The two methods used have not lead yet to a successful attack on the full compression function and RIPEMD is still holding up.

9 Appendix

Description of the RIPEMD RIPEMD^L and RIPEMD^R Compression and Hash Functions

Define the ρ and σ permutations of [0..15] give by Table 4 hereafter

Table 4. Permutations for rounds 2 and 3

Word	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ρ	9	3	13	7	1	11	5	0	15	10	4	14	8	2	12	6
σ	10	7	2	0	3	14	11	9	6	4	1	12	12	13	8	5

Define register R_k ($0 \leq k \leq 47$) as the 32-bit register value computed at step k. R_k is also denoted A_k when $k = 0 \bmod 4$, D_k when $k = 1 \bmod 4$, C_k when $k = 2 \bmod 4$ and B_k when $k = 3 \bmod 4$ as far as the left line is concerned. The same definitions apply for the right line and for R'_k, A'_k, D'_k, C'_k and B'_k.

Start with the initial value $IV = (IV_A, IV_B, IV_C, IV_D)$. For the first iteration, start with $IV = IV0 = (IV_A, IV_B, IV_C, IV_D)$, where $IV_A = 0x67452301$; $IV_B = 0xefcdab89$; $IV_C = 0x98badcfe$; $IV_D = 0x10325476$.

Copy IV_A, IV_B, IV_C, IV_D into the registers A, B, C, D for the left line and $A'B'C'D'$ for the right line. These registers correspond to steps -4, -3, -2 and -1.

Define the constants $K_0 = 0x0$, $K_1 = 0x5a827999$, $K_2 = 0x6ed9eba1$ for the left line and $K'_0 = 0x50a28be6$, $K'_1 = 0x0$, $K'_2 = 0x5c4dd124$ for the right line.

Define $F(x, y, z) = (x \wedge y) \vee ((\neg x) \wedge z)$, $G(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (z \wedge y)$ and $H(x, y, z) = (x \oplus y \oplus z)$ functions used in rounds 1, 2 and 3 respectively.

Define the word X and the shift for each step given by Table 5 hereafter.

Compute for ($i = 0$; $i < 48$;$i = i + 1$):

$$R_i = (R_{i-4} + f_r(R_{i-1}, R_{i-2}, R_{i-3}) + X_{\varphi(i \bmod 16)} + K_0)^{<<s_i}$$
$$R'_i = (R'_{i-4} + f_r(R'_{i-1}, R'_{i-2}, R'_{i-3}) + X_{\varphi(i \bmod 16)} + K'_0)^{<<s_i}$$

Where $r = 0$ when $i \in [0..15]$, $r = 1$ when $i \in [16..31]$ and $r = 2$ when $i \in [32..47]$ and $f_0 = F$, $f_1 = G$ and $f_2 = H$.

Table 5. Rotations for each step

step	X	s	X	s	X	s	X	s	step	X	s	X	s	X	s	X	s	step	X	s	X	s	X	s	X	s
0 to 3	0	11	1	14	2	15	3	12	16 to 19	7	7	4	6	13	8	1	13	32 to 35	3	11	10	13	2	14	4	7
4 to 7	4	5	5	8	6	7	7	9	20 to 23	10	11	6	9	15	7	3	15	36 to 39	9	14	15	9	8	13	1	15
8 to 11	8	11	9	13	10	14	11	15	24 to 27	12	7	0	12	9	15	5	9	40 to 43	14	6	7	8	0	13	6	6
12 to 15	12	6	13	7	14	9	15	8	28 to 31	14	7	2	11	11	13	8	12	44 to 47	11	12	13	5	5	7	12	5

Finally compute the compression function output AA, BB, CC, DD as follows:

For the RIPEMD compression function: $AA = IVB + C + D'$,
$BB = IVC + D + A'$, $CC = IVD + A + B'$, $DD = IVA + B + C'$
For the RIPEMD^L compression function: $AA = IVA + A$,
$BB = IVB + B$, $CC = IVC + C$, $DD = IVD + D$.
For the RIPEMD^R compression function: $AA = IVA + A'$,
$BB = IVB + B'$, $CC = IVC + C'$, $DD = IVD + D'$.

References

1. F.Chabaud and A.Joux. Differential Collisions in SHA-0. extended abstract. In CRYPTO'98, LNCS 1462, pp 56–71, 1998.
2. B.den Boer and A.Bosselaers. An attack on the last two rounds of MD4. In Advances in Cryptology - Crypto'91 pages 194-203 LCNS 576 Springer-Verlag 1992.
3. H.Dobbertin. Cryptanalysis of MD4. In Journal of Cryptology vol.11 n.4 Autumn 1998.
4. H.Dobbertin. Cryptanalysis of MD5 Compress. Presented at the rump session of Eurocrypt '96, May 14, 1996.
5. H.Dobbertin. Ripemd with two round compress function is not collision-free. In Journal of Cryptology vol.10 n.1, winter 1997.
6. H.Dobbertin, A.Bosselaers and B.Preneel. RIPEMD-160: a strenghened version of RIPEMD. April 1996.
ftp.esat.kuleuven.ac.be/pub/COSIC/bossselae/ripemd.
7. National Institute of Standards and Technology (NIST) FIPS Publication 180-1: secure Hash Standard. April 1994.
8. RIPE. Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040). In LNCS 1007 Springer-Verlag 1995.
9. R.L.Rivest. The MD4 message digest algorithm. In Advances in Cryptology - Crypto'90 pages 303-311 Springer-Verlag 1991.
10. R.L.Rivest. RFC1321: The MD5 message digest algorithm. M.I.T. Laboratory for Computer Science and RSA Data Security, Inc., April 1992.
11. S.Vaudenay. On the need for multipermutations: Cryptanalysis of MD4 and SAFER. In FSE, LCNS 1008, pages 286-297 Springer-Verlag 1995.

New Constructions of Resilient Boolean Functions with Maximal Nonlinearity

Yuriy Tarannikov

Mech. & Math. Department
Moscow State University
119899 Moscow, Russia
yutaran@mech.math.msu.su, taran@vertex.inria.msu.ru

Abstract. In this paper we develop a technique that allows to obtain new effective constructions of highly resilient Boolean functions with high nonlinearity. In particular, we prove that the upper bound $2^{n-1} - 2^{m+1}$ on nonlinearity of m-resilient n-variable Boolean functions is achieved for $0.6n - 1 \leq m \leq n - 2$.

Keywords. Stream cipher, Boolean function, nonlinear combining function, correlation-immunity, resiliency, nonlinearity.

1 Introduction

One of the most general types of stream cipher systems is several Linear Feedback Shift Registers (LFSRs) combined by nonlinear Boolean function. This function must satisfy certain criteria to resist different attacks (in particular, correlation attacks suggested by Siegenthaler [18] and different types of linear attacks). The following factors are considered as important properties of Boolean functions for using in stream cipher applications.

1. *Balancedness.* A Boolean function must output zeroes and ones with the same probabilities.

2. Good *correlation-immunity* (of order m). The output of Boolean function must be statistically independent of combination of any m its inputs. A balanced correlation-immune of order m Boolean function is called *m-resilient.*

3. Good *nonlinearity.* The Boolean function must be at the sufficiently high distance from any affine function.

Other important factors are large algebraic degree and simple implementation in hardware.

The variety of criteria and complicated trade-offs between them caused the next approach: to fix one or two parameters and try to optimize others. The most general model is when researchers fix the parameters n (number of variables) and m (order of correlation-immunity) and try to optimize some other cryptographically important parameters. Here we can call the works [16], [2], [7], [4] [8], [9], [10], [12], [19], [20], [22].

The present paper continues the investigations in this direction and gives new results. In Section 2 we give preliminary concepts and notions. In Section 3 we

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 66–77, 2002.

give a brief review of the investigations on the problem of maximal nonlinearity for n-variable m-resilient Boolean function. In Section 4 we discuss a concept of a linear and a pair of quasilinear variables which works in the following sections. In Section 5 we present our main construction method. This method is a generalization of a method described in [19] and [20]. This method allows to construct recursively the functions with good cryptographic properties using the functions with good cryptographic properties and smaller number of variables. The method is based on the existence of a *proper* matrix with prescribed properties. In Section 6 we give some examples of proper matrices and obtain new results on the maximal nonlinearity $nlmax(n,l)$ of m-resilient functions on V^n. Namely, we prove that $nlmax(n,m) = 2^{n-1} - 2^{m+1}$ for $\frac{5n-14}{8} \leq m \leq n-2$ and for $0.6n-1 \leq m \leq n-2$. In Section 7 we give some remarks on the combinatorial problem connected with proper matrices and give a geometrical interpretations of proper matrices.

2 Preliminary Concepts and Notions

We consider V^n, the vector space of n tuples of elements from $GF(2)$. A *Boolean function* is a function from V^n to $GF(2)$. The *weight* $wt(f)$ of a function f on V^n is the number of vectors x on V^n such that $f(x) = 1$. A function f is said to be *balanced* if $wt(f) = wt(f \oplus 1)$. Obviously, if a function f on V^n is balanced then $wt(f) = 2^{n-1}$. A *subfunction* of the Boolean function f is a function f' obtained by substitution some constants for some variables in f. If we substitute in the function f the constants $\sigma_{i_1}, \ldots, \sigma_{i_s}$ for the variables $x_{i_1}, \ldots, x_{i_s}$ respectively then the obtained subfunction is denoted by $f^{\sigma_{i_1},\ldots,\sigma_{i_s}}_{x_{i_1},\ldots,x_{i_s}}$. If a variable x_i is not substituted by constant then x_i is called a *free* variable for f'.

It is well known that a function f on V^n can be uniquely represented by a polynomial on $GF(2)$ whose degree is at most n. Namely,

$$f(x_1, \ldots, x_n) = \bigoplus_{(a_1,\ldots,a_n)\in V^n} g(a_1, \ldots, a_n) x_1^{a_1} \ldots x_n^{a_n}$$

where g is also a function on V^n. This polynomial representation of f is called the *algebraic normal form* (briefly, ANF) of the function and each $x_1^{a_1} \ldots x_n^{a_n}$ is called a *term* in ANF of f. The *algebraic degree* of f, denoted by $\deg(f)$, is defined as the number of variables in the longest term of f. The *algebraic degree of variable* x_i in f, denoted by $\deg(f, x_i)$, is the number of variables in the longest term of f that contains x_i. If $\deg(f, x_i) = 1$, we say that x_i is a *linear* variable in f. The term of length 1 is called a *linear* term. If $\deg(f) \leq 1$ then f is called an *affine* function.

The *Hamming distance* $d(x', x'')$ between two vectors x' and x'' is the number of components where vectors x' and x'' differ. For two Boolean functions f_1 and f_2 on V^n, we define the distance between f_1 and f_2 by $d(f_1, f_2) = \#\{x \in V^n | f_1(x) \neq f_2(x)\}$. The minimum distance between f and the set of all affine functions is called the *nonlinearity* of f and denoted by $nl(f)$.

A Boolean function f on V^n is said to be *correlation-immune of order* m, with $1 \le m \le n$, if the output of f and any m input variables are statistically independent. This concept was introduced by Siegenthaler [17]. In equivalent non-probabilistic formulation the Boolean function f is called correlation-immune of order m if $wt(f') = wt(f)/2^m$ for any its subfunction f' of $n-m$ variables. A balanced mth order correlation immune function is called an *m-resilient* function. In other words the Boolean function f is called m-resilient if $wt(f') = 2^{n-m-1}$ for any its subfunction f' of $n-m$ variables. From this point of view we can consider formally any balanced Boolean function as 0-resilient (this convention is accepted in [1], [9], [12]) and an arbitrary Boolean function as (-1)-resilient. The concept of an m-resilient function was introduced in [3].

Siegenthaler's Inequality [17] states that if the function f is a correlation-immune function of order m then $\deg(f) \le n - m$. Moreover, if f is an m-resilient, $m \le n-2$, then $\deg(f) \le n - m - 1$.

The next lemma is well-known.

Lemma 1. *Let $f(x_1, \ldots, x_n)$ be a Boolean function represented in the form*

$$f(x_1, \ldots, x_n) = \bigoplus_{(\sigma_1, \ldots, \sigma_l)} (x_1 \oplus \sigma_1) \ldots (x_l \oplus \sigma_l) f(\sigma_1 \oplus 1, \ldots, \sigma_l \oplus 1, x_{l+1}, \ldots, x_n).$$

Suppose that all 2^l subfunctions $f(\sigma_1 \oplus 1, \ldots, \sigma_l \oplus 1, x_{l+1}, \ldots, x_n)$ are m-resilient. Then the function f is an m-resilient too.

The Lemma 1 was proved in a lot of papers including (for $l = 1$) the pioneering paper of Siegenthaler (Theorem 2 in [17]). General case follows immediately from the case $l = 1$.

3 The Problem of Maximal Nonlinearity for Resilient Functions

Let m and n be integers, $-1 \le m \le n$. Denote by $nlmax(n, m)$ the maximal possible nonlinearity of m-resilient Boolean function on V^n. It is well-known that the nonlinearity of a Boolean function does not exceed $2^{n-1} - 2^{\frac{n}{2}-1}$ [15]. Thus, $nlmax(n, -1) \le 2^{n-1} - 2^{\frac{n}{2}-1}$. This value can be achieved only for even n. The functions with such nonlinearity are called *bent functions.* Thus, for even n we have $nlmax(n, -1) = 2^{n-1} - 2^{\frac{n}{2}-1}$. It is known [13,14,7] that for odd n, $n \le 7$, $nlmax(n, -1) = 2^{n-1} - 2^{(n-1)/2}$, and for odd n, $n \ge 15$, the inequality $nlmax(n, -1) > 2^{n-1} - 2^{(n-1)/2}$ holds. Bent functions are nonbalanced always, so, for balanced (0-resilient) n-variable function f we have $nl(f) < 2^{n-1} - 2^{\frac{n}{2}-1}$, and $nlmax(n, m) < 2^{n-1} - 2^{\frac{n}{2}-1}$ for $m \ge 0$. If f is n-variable m-resilient function, $m \ge n-2$, then by Siegenthaler's Inequality [17] $\deg(f) \le 1$, so $nlmax(n, m) = 0$. For some small values of parameters n and m exact values of maximal nonlinearity are known. The latest collection of such values is given in [10]. The upper bound $nlmax(n, m) \le 2^{n-1} - 2^{m+1}$ for $m \le n-1$ was proven independently in [10], [19] and [23], all three these manuscripts were submitted

to Crypto 2000 although only the first was accepted). In [19] and [20] an effective construction of m-resilient function on V^n with nonlinearity $2^{n-1} - 2^{m+1}$ was given. This construction gives that $nlmax(n,m) = 2^{n-1} - 2^{m+1}$ for $\frac{2n-7}{3} \leq m \leq n-2$. The construction of [19] and [20] was slightly modified in [11]. The last modification follows that $nlmax(n,m) = 2^{n-1} - 2^{m+1}$ for $\frac{2n-8}{3} \leq m \leq n-2$. In [10] it was proved that the nonlinearity of m-resilient function on V^n is divided by 2^{m+1}. Also in [10] it was proved that $nlmax(n,m) \leq 2^{n-1} - 2^{n/2-1} - 2^{m+1}$ for $m < (n/2) - 2$ (for the case of odd n a bit more strong bound was given).

4 On Linear and Quasilinear Variables

In this section we recall the concepts of linear and quasilinear variables. The last concept was introduced in [19].

Recall that a variable x_i is called *a linear* for a function $f = f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$ if $\deg(f, x_i) = 1$. Also we say that a function f depends on a variable x_i *linearly*. If a variable x_i is linear for a function f we can represent f in the form

$$f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) = g(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \oplus x_i.$$

Other equivalent definition of a linear variable is that a variable x_i is linear for a function f if $f(x') \neq f(x'')$ for any two vectors x' and x'' that differ only in ith component. By analogy with the last definition we give a new definition for a pair of quasilinear variables.

Definition 1. *We say that a Boolean function $f = f(x_1, \ldots, x_n)$ depends on a pair of its variables (x_i, x_j)* quasilinearly *if $f(x') \neq f(x'')$ for any two vectors x' and x'' of length n that differ only in ith and jth components. A pair (x_i, x_j) in this case is called a* pair of quasilinear variables *in f.*

The proof of the next lemma is given in [19] and [20].

Lemma 2. *Let $f(x_1, \ldots, x_n)$ be a Boolean function. Then (x_i, x_j), $i < j$, is a pair of quasilinear variables in f iff f can be represented in the form*

$$f(x_1, \ldots, x_n) = g(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n, x_i \oplus x_j) \oplus x_i. \quad (1)$$

The next lemmas are obvious.

Lemma 3. *Let $f(x_1, \ldots, x_n)$ be a Boolean function. If f depends on some variable x_i linearly then f is balanced.*

Lemma 4. *$f(x_1, \ldots, x_n)$ be a Boolean function. If f depends on some variables $x_{i_1}, x_{i_2}, \ldots, x_{i_s}$ linearly then f is $(s-1)$-resilient.*

Note that Lemma 4 agrees with our assumption that a balanced function is 0-resilient, and an arbitrary Boolean function is (-1)-resilient. (In the last case $s = 0$.)

Lemma 5. *Let $f(x_1, \ldots, x_n)$ be a Boolean function. If f depends on some pair of variables (x_i, x_j) quasilinearly then f is balanced.*

Lemma 6. *Let $f(x_1, \ldots, x_n, x_{n+1}) = f(x_1, \ldots, x_n) \oplus cx_{n+1}$ where $c \in \{0, 1\}$. Then $nl(f) = 2nl(g)$.*

Lemma 7. *Let $f(x_1, \ldots, x_n)$ be a Boolean function on V^n and f depends on some pair of variables (x_i, x_j) quasilinearly. Then $nl(f) = 2nl(g)$ where g is a function used in the representation of f in Lemma 2.*

Lemma 8. *Let f_1 and f_2 be two Boolean functions on V^n. Moreover, there exist variables x_i and x_j such that f_1 depends on a pair of variables (x_i, x_j) quasilinearly whereas f_2 depends on the variables x_i and x_j linearly. Let l be an arbitrary affine function on V^n. Then at least one of two functions $f_1 \oplus l$ and $f_2 \oplus l$ is balanced.*

Proof. Let $l = \bigoplus_{r=1}^{n} u_r x_r \oplus u_0$, $u_r \in \{0, 1\}$, $r = 0, 1, \ldots, n$. If $u_i = 0$ (correspondently, $u_j = 0$) then $f_2 \oplus l$ depends on the variable x_i (x_j) linearly, therefore the function $f_2 \oplus l$ is balanced. The remained case is $u_i = u_j = 1$. But here it is easy to see that the function $f_1 \oplus l$ depends on a pair of variables (x_i, x_j) quasilinearly, therefore $f_1 \oplus l$ is balanced. □

5 A Method of Constructing

Suppose that $f_0, f_1, \ldots, f_{2^k-1}$ are Boolean functions on V^n. We denote f_r also as $f_{\sigma_1 \ldots \sigma_k}$ where $\sigma_1 \ldots \sigma_k$ is a binary representation of the number r. Suppose that $c = (c_1, \ldots, c_k)$ is an arbitrary binary vector. Put $s = \sum_{i=1}^{k} c_i$. We denote $X = \{x_i \,|\, i = 1, \ldots, n\}$, $Y = \{y_i \,|\, i = 1, \ldots, k\}$, $Z = \{z_i \,|\, c_i = 1,\ i = 1, \ldots, k\}$ (The set Z contains the variables z_i only for i such that $c_i = 1$). We define

$$f(X, Y, Z) = \left(\bigoplus_{(\sigma_1, \ldots, \sigma_k) \in V^k} \left(\prod_{i=1}^{k} (y_i \oplus c_i z_i \oplus \sigma_i) \right) f_{\sigma_1 \ldots \sigma_k}(X) \right) \oplus \bigoplus_{i=1}^{k} c_i z_i. \quad (2)$$

By construction the function f in (2) depends on $n + k + s$ variables. Below we formulate some properties of construction (2).

Remark. Some details of construction (2) can be understood more easily if we put $c_1 = \ldots = c_s = 1$, $c_{s+1} = \ldots = c_k = 0$. But for an effective implementation it is important in some cases to vary the vector c.

Lemma 9. *Suppose that all 2^k Boolean functions f_0, f_1, …, f_{2^k-1} in (2) are m-resilient. Then the function $f(X, Y, Z)$ is $(m + s)$-resilient.*

Proof. Substitute in (2) arbitrary $m+s$ constants for arbitrary $m+s$ variables. We obtain some $(n + k - m)$-variable subfunction f'. If $c_i = 1$ for some i and

if both variables y_i and z_i are free in f' then the pair of variables (y_i, z_i) is a quasilinear pair in f' therefore by Lemma 5 the subfunction f' is balanced). Thus, we can assume that for each i such that $c_i = 1$ at least one of two variables y_i and z_i is substituted by constant. Then at most m variables from X are substituted by constants in (2). All functions $f_0, f_1, \ldots, f_{2^k-1}$ are balanced, therefore the function $f(X, Y, Z)$ is balanced too. We have proved that an arbitrary $(n+k-m)$-variable subfunction of $f(X, Y, Z)$ is balanced. □

Lemma 10. *Suppose that the nonlinearity of all 2^k Boolean functions f_0, f_1, $\ldots$, f_{2^k-1} in (2) is at least N_0. Moreover, for any two functions f_{r_1} and f_{r_2}, $0 \le r_1 \ne r_2 \le 2^k - 1$, there exists a pair of variables (x_i, x_j) such that one of these two functions, say f_{r_1} depends linearly on the variables x_i and x_j whereas another function f_{r_2} depends quasilinearly on the pair (x_i, x_j). Then $nl(f) \ge 2^s(2^{n-1}(2^k - 1) + N_0)$.*

Proof. It is obvious that if we replace $c_i = 0$ by $c_i = 1$ then we multiplate the nonlinearity by 2 (adding new variable). Thus we can assume that $s = 0$. Consider an arbitrary affine function l. Denote $l_r = l_{y_1, \ldots, y_k}^{\sigma_1 \oplus 1, \ldots, \sigma_k \oplus 1}$ where $\sigma_1 \ldots \sigma_k$ is a binary representation of the number r. Note that for any $r = 0, \ldots, 2^k - 1$, we have $l_r = l_0$ or $l_r = l_0 \oplus 1$. Then $d(f, l) = \sum_{r=0}^{2^k-1} d(f_r, l_r)$. By Lemma 8 and the hypothesis of this lemma we have that $d(f_r, l_r) \ne 2^{n-1}$ for at most one value of r. Thus $d(f, l) \ge 2^{n-1}(2^k - 1) + N_0$. An affine function l was chosen arbitrary. Therefore $nl(f) \ge 2^{n-1}(2^k - 1) + N_0$. □

The construction (2) is a generalization of the construction in [19] and [20] where only the case $k = 1$ is considered.

The problem is to find the functions $f_0, f_1, \ldots, f_{2^k-1}$ with desirable distribution of linear and quasilinear variables. Below we give some approach that allows to construct such systems of functions.

Definition 2. *Let $B = (b_{ij})$ be $(2^k \times p)$ matrix of 2^k rows and p columns with entries from the set $\{1, 2, *\}$. Let k_0 and t be positive integers. We assume that*

(i) for every two rows i_1 and i_2 there exists a column j such that $b_{i_1 j} = 1$, $b_{i_2 j} = 2$ or $b_{i_1 j} = 2$, $b_{i_2 j} = 1$.

(ii) for every row i the inequality $\sum_{j=1}^{p} b_{ij} \le t$ holds (a sign $*$ does not give an influence to these sums).

(iii) in every row the number of ones does not exceed k_0.

If the matrix B satisfies all properties (i), (ii), (iii) we say that B is a proper *(k_0, k, p, t)-matrix.*

Definition 3. *Let F be a set of Boolean functions such that for every s, $0 \le s \le k$, the set F contains an $(m+s)$-resilient function on V^{n+s} with nonlinearity at least $2^s(2^{n-1} - 2^{m+\lambda})$ (λ is not necessary integer). Moreover, we assume that each f_s contains s disjoint pairs of quasilinear variables. Then we say that F is a $S_{n,m,k,\lambda}$-*system *of Boolean functions.*

Remark. To provide an existence of a $S_{n,m,k,\lambda}$-system of Boolean functions it is sufficient to have only one $(m+k)$-resilient function f on V^{n+k} with nonlinearity at least $2^k(2^{n-1}-2^{m+\lambda})$ that contains k disjoint pairs of quasilinear variables. All other necessary functions of $S_{n,m,k,\lambda}$-system can be obtained from f by substitutions of constants for the variables from different disjoint pairs of quasilinear variables. But note that the last way is not effective from the implementation point of view.

Lemma 11. *There exists an $S_{2,-1,2,1}$-system of Boolean functions.*

Proof. Put $f'_0 = x_1x_2$, $f'_1 = (x_1 \oplus x_2)x_3 \oplus x_1$, $f'_2 = (x_1 \oplus x_2)(x_3 \oplus x_4) \oplus x_1 \oplus x_3$. It is easy to verify that f'_s, $s = 0, 1, 2$, is a $(-1+s)$-resilient function on V^{2+s} with nonlinearity $2^s(2^{2-1} - 2^{-1+1})$, moreover, f'_s contains s disjoint pairs of quasilinear variables. □

Theorem 1. *Suppose that there exists an $S_{n,m,k_0,\lambda}$-system of Boolean functions F and there exists a proper (k_0, k, p, t)-matrix B, $n \geq 2p - t$. Then there exists an $S_{n+k+t,m+t,k,\lambda}$-system of Boolean functions.*

Proof. Consider the ith row of the matrix B, $i = 0, 1, \ldots, 2^k - 1$. Suppose that this row contains $s = s(i)$ ones. The matrix B is a proper, therefore $s \leq k_0$, $s \leq t$. By assumption there exists an $(m+s)$-resilient function f'_i on V^{n+s} that contains s disjoint pairs of quasilinear variables with nonlinearity at least $2^s(2^{n-1}-2^{m+\lambda})$. Add $t - s$ new linear variables to the function f'_i. As a result we obtain the function f''_i on V^{n+t}. It is easy to see that the function f''_i is an $(m+t)$-resilient function with nonlinearity at least $2^t(2^{n-1} - 2^{m+\lambda})$, moreover f''_i contains s disjoint pairs of quasilinear variables and besides $t-s$ linear variables. Note that by the property (ii) of a proper matrix the value $t-s$ is not less than the number of 2's in ith row of B multiplied by 2. By this way we construct the functions f''_i on V^{n+t} for every i, $i = 0, 1, \ldots, 2^k - 1$. By assumption $n+t \geq 2p$. Next, for every i, $i = 0, 1, \ldots, 2^k - 1$, we permute the variables in $f''_i(x_1, \ldots, x_{n+t})$ obtaining the function f_i such that the function f_i depends on a pair of variables (x_{2j-1}, x_{2j}) quasilinearly if $b_{ij} = 1$, and the function f_i depends on the variables x_{2j-1} and x_{2j} linearly if $b_{ij} = 2$. By the arguments given above we have sufficient numbers of quasilinear and linear variables for this procedure. Now we are ready to apply the construction (2). By means of this construction varying the number of ones in the vector $(c_1, \ldots, c_k)$ we obtain the functions $f(X, Y, Z_s)$, $s = 0, 1, \ldots, k$. The function $f(X, Y, Z_s)$ by Lemmas 9 and 10 is an $(m+t+s)$-resilient function on $V^{n+k+t+s}$ with the nonlinearity at least $2^s(2^{n+k+t-1} - 2^{m+t+\lambda})$. Moreover, the function $f(X, Y, Z_s)$ contains s disjoint pairs of quasilinear variables. Thus, we have constructed an $S_{n+k+t,m+t,k,\lambda}$-system of Boolean functions. □

An application of the construction given in Theorem 1 we denote by

$$S_{n,m,k_0,\lambda}T_{k_0,k,p,t} = S_{n+k+t,m+t,k,\lambda}.$$

If we add new linear variable to an m-resilient function f on V^n then we obtain $(m+1)$-resilient function f on V^{n+1} with nonlinearity $2nl(f)$. We denote this procedure by

$$S_{n,m,0,\lambda}T_{0,0,0,1} = S_{n+1,m+1,0,\lambda}.$$

6 Examples of Proper Matrices Effective for Our Construction and New Resilient Boolean Functions with Maximal Nonlinearity

At first, we give some examples of proper matrices effective for the construction of Boolean functions with good combination of parameters. We denote a proper (k_0, k, p, t)-matrix by $B_{k_0,k,p,t}$.

$$B_{1,1,1,2} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \; B_{2,2,2,4} = \begin{pmatrix} 2 & 2 \\ 2 & 1 \\ 1 & 2 \\ 1 & 1 \end{pmatrix}, \; B_{3,2,3,3} = \begin{pmatrix} 2 & 1 & * \\ * & 2 & 1 \\ 1 & * & 2 \\ 1 & 1 & 1 \end{pmatrix},$$

$$B_{2,3,5,6} = \begin{pmatrix} 2 & 2 & 1 & 1 & * \\ 2 & 1 & 1 & 2 & * \\ 2 & 1 & * & 1 & 2 \\ 2 & 1 & 2 & * & 1 \\ 1 & 1 & * & 2 & 2 \\ 1 & 2 & 1 & * & 2 \\ 1 & * & 2 & 1 & 2 \\ 1 & * & 2 & 2 & 1 \end{pmatrix}, \; B_{3,3,4,5} = \begin{pmatrix} * & 1 & 2 & 2 \\ 2 & * & 1 & 2 \\ 2 & 2 & * & 1 \\ 1 & 2 & 2 & * \\ 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix},$$

$$B_{2,4,7,8} = \begin{pmatrix} 1 & 1 & * & * & 2 & 2 & 2 \\ * & 2 & 1 & 1 & * & 2 & 2 \\ * & * & 2 & 2 & 1 & 1 & 2 \\ 1 & * & * & 2 & 2 & 2 & 1 \\ 2 & 1 & 1 & * & * & 2 & 2 \\ * & * & 2 & 1 & 1 & 2 & 2 \\ * & * & 2 & 2 & 2 & 1 & 1 \\ 1 & 2 & 2 & 1 & 2 & * & * \\ * & 1 & 2 & 2 & 1 & 2 & * \\ * & * & 1 & 2 & 2 & 1 & 2 \\ 2 & * & * & 1 & 2 & 2 & 1 \\ 1 & 2 & * & 2 & 1 & 2 & * \\ * & 1 & 2 & * & 2 & 1 & 2 \\ 2 & * & 1 & 2 & * & 2 & 1 \\ 2 & 2 & * & 1 & * & 1 & 2 \\ 2 & 2 & 2 & * & 1 & * & 1 \end{pmatrix}, B_{4,4,6,6} = \begin{pmatrix} 2 & 2 & 2 & * & * & * \\ 1 & 2 & * & 1 & 2 & * \\ 1 & 2 & * & * & 1 & 2 \\ 1 & 2 & * & 2 & * & 1 \\ * & 1 & 2 & 1 & 2 & * \\ * & 1 & 2 & * & 1 & 2 \\ * & 1 & 2 & 2 & * & 1 \\ 2 & * & 1 & 1 & 2 & * \\ 2 & * & 1 & * & 1 & 2 \\ 2 & * & 1 & 2 & * & 1 \\ 2 & * & 1 & 1 & 1 & 1 \\ 1 & 2 & * & 1 & 1 & 1 \\ * & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & * & 1 \\ 1 & 1 & 1 & 1 & 2 & * \\ 1 & 1 & 1 & * & 1 & 2 \end{pmatrix}.$$

It is easy to verify that all matrices given above are proper matrices with correspondent parameters.

The simplest example of a proper matrix is the matrix $B_{1,1,1,2}$. If $\frac{2n-7}{3} \leq m \leq n-3$ then the numbers n and m can be represented in the form $n = 3r + s + 2$, $m = 2r + s - 1$, where r and s are nonnegative integers (an existence of this representation as well as an existence of the representations in Theorems 2 and 3 can be proved by the arguments from the elementary arithmetic). By Lemma 11 there exists a system $S_{2,-1,2,1}$. We apply

$$S_{2,-1,2,1}(T_{1,1,1,2})^r(T_{0,0,0,1})^s = S_{n,m,0,1}.$$

Therefore $nlmax(n,m) \geq 2^{n-1} - 2^{m+1}$ for $m \geq \frac{2n-7}{3}$. In Section 3 it was pointed out that $nlmax(n,m) \leq 2^{n-1} - 2^{m+1}$ for $m \leq n-2$. Therefore $nlmax(n,m) = 2^{n-1} - 2^{m+1}$ for $\frac{2n-7}{3} \leq m \leq n-2$. The above construction was given in [19] and [20].

Theorem 2. *$nlmax(n,m) = 2^{n-1} - 2^{m+1}$ for $\frac{5n-14}{8} \leq m \leq n-2$.*

Proof. Let n, m be integers. Note that $\lceil \frac{5n-14}{8} \rceil \geq \lceil \frac{2n-7}{3} \rceil$ for $n < 17$. If $n \geq 17$, $m > n-8$, then $m \geq \frac{2n-7}{3}$. If $n \geq 17$, $\frac{5n-13}{8} \leq m \leq n-8$, then the numbers n and m can be represented in the form $n = 8r_1 + 3r_2 + s + 17$, $m = 5r_1 + 2r_2 + s + 9$, where r_1, r_2 and s are nonnegative integers. We apply

$$S_{2,-1,2,1}T_{2,2,2,4}T_{2,3,5,6}(T_{3,3,4,5})^{r_1}(T_{1,1,1,2})^{r_2}(T_{0,0,0,1})^{s} = S_{n,m,0,1}.$$

If $n \geq 17$, $\frac{5n-14}{8} = m$, then the numbers n and m can be represented in the form $n = 8r + 22$, $m = 5r + 12$, where r is nonnegative integer. In this case we apply

$$S_{2,-1,2,1}T_{2,2,2,4}T_{2,3,5,6}(T_{3,3,4,5})^{r}T_{3,2,3,3} = S_{n,m,2,1}.$$

□

Theorem 3. *$nlmax(n,m) = 2^{n-1} - 2^{m+1}$ for $0.6n - 1 \leq m \leq n-2$.*

Proof. Let n, m be integers. Note that $0.6n - 1 \geq \frac{2n-7}{3}$ for $n \leq 20$. If $n \geq 20$, $m > n-9$, then $m \geq \frac{2n-7}{3}$. If $n \geq 20$, $0.6n - 1 \leq m \leq n-9$, excepting the case $m = 0.6n - 1$, $n \equiv 5 \pmod{10}$, then the numbers n and m can be represented in the form $n = 10r_1 + 8r_2 + 3r_3 + s + 20$, $m = 6r_1 + 5r_2 + 2r_3 + s + 11$, where r_1, r_2,r_3 and s are nonnegative integers. We apply

$$S_{2,-1,2,1}T_{2,2,2,4}T_{2,4,7,8}(T_{4,4,6,6})^{r_1}(T_{3,3,4,5})^{r_2}(T_{1,1,1,2})^{r_3}(T_{0,0,0,1})^{s} = S_{n,m,0,1}.$$

In the case $n = 10r + 25$, $m = 6r + 14$, where r is a nonnegative integer we apply

$$S_{2,-1,2,1}T_{2,2,2,4}T_{2,4,7,8}(T_{4,4,6,6})^{r}T_{3,2,3,3} = S_{n,m,2,1}.$$

□

Note that the best previous result [11] was that $nlmax(n,m) = 2^{n-1} - 2^{m+1}$ for $\frac{2n-8}{3} \leq m \leq n-2$. Since $\frac{5n-14}{8} < \frac{2n-8}{3}$ for $n > 22$ it is easy to check that our constructions provide the best known result beginning with $n = 25$.

7 Some Remarks on Combinatorial Problem and Geometrical Interpretations

If there exists a proper (k,k,p,t)-matrix then using the technique described in the previous section we can prove that $nlmax(n,m) = 2^{n-1} - 2^{m+1}$ for $m > \frac{t}{t+k}n - c'$ where c' is a some constant. Therefore we are interesting to find a proper (k,k,p,t)-matrix where the ratio $\frac{t}{k}$ is as small as possible.

For given positive integer k we denote by $t(k)$ the minimal positive integer t such that for some p there exists a proper (k,k,p,t)-matrix. It is clear that we can consider only matrices without all-$*$ columns. Then obviously $p \le t \cdot 2^k$. There exists a proper $(k,k,k,2k)$-matrix (all rows are different and without $*$). Thus, to find $t(k)$ it is sufficiently to consider only a finite set of matrices.

Proposition 1. *Let k_1 and k_2 be positive integers. Then $t(k_1+k_2) \le t(k_1) + t(k_2)$.*

Proof. By definition for some p_1 and p_2 there exist a proper $(k_1,k_1,p_1,t(k_1))$-matrix B' and a proper $(k_2,k_2,p_2,t(k_2))$-matrix B''. Compose a $(2^{k_1+k_2} \times (p_1+p_2))$ matrix B where the rows of B are all possible concatenations of rows of matrices B' and B''. It is easy to see that B is a proper $(k_1+k_2, k_1+k_2, p_1+p_2, t(k_1)+t(k_2))$-matrix. Therefore $t(k_1+k_2) \le t(k_1)+t(k_2)$. □

It is quite obvious that

Proposition 2. $t(k) \ge k$.

Propositions 1 and 2 follow that there exists the limit $\lim\limits_{k\to\infty} \frac{t(k)}{k}$.

A proper (k_0,k,p,t)-matrix B can be interpreted as a collection of 2^k disjoint subcubes in Boolean cube $\{1,2\}^p$. Indeed, a row of B can be interpreted as a subcube where the components with $*$ are free whereas the components with 1 or 2 are substituted by correspondent constant. We illustrate this at the example of the matrix $B_{3,3,4,5}$:

row of B_{3345}	points of a subcube
$*122$	$\{(1,1,2,2),(2,1,2,2)\}$
$2*12$	$\{(2,1,1,2),(2,2,1,2)\}$
$22*1$	$\{(2,2,1,1),(2,2,2,1)\}$
$122*$	$\{(1,2,2,1),(1,2,2,2)\}$
2111	$\{(2,1,1,1)\}$
1211	$\{(1,2,1,1)\}$
1121	$\{(1,1,2,1)\}$
1112	$\{(1,1,1,2)\}$

The property (i) of a proper matrix provides that subcubes are disjoint. The properties (ii) and (iii) characterize the location of subcubes in a cube and the size of subcubes.

Estimating the numbers of points at different levels of Boolean cube that belong to some disjoint subcubes we are able to prove that

Proposition 3. $t(1)=2$, $t(2)=4$, $t(3)=5$, $t(4)=6$, $t(5)=8$, $t(6)=9$, $t(7)=11$, $t(8)=12$, $t(10)=15$.

Already after FSE 2001 Fedorova and Tarannikov analysing a preliminary version of this paper [21] proved in [5] and [6] that there do not exist proper (k_0,k,p,t)-matrices for $\frac{t}{t+k} < \frac{1}{\log_2(\sqrt{5}+1)} = 0.5902\ldots$ It follows that by means

of only the method (2) with using of proper matrices it is impossible to construct m-resilient function on V^n with maximal possible nonlinearity $2^{n-1} - 2^{m+1}$ for $m < 0.5902\ldots n + O(1)$. At the same time in [5] and [6] it is proved that $\lim\limits_{k\to\infty} \frac{t(k)}{k} = \frac{1}{\log_2(\sqrt{5}+1)} = 0.5902\ldots$ and it is constructed at infinite sequence of such functions with $m = 0.5902\ldots n + O(\log_2 n)$. Nevertheless the smallest parameters given in [5] and [6] that improve our bounds are $n = 172$, $m = 102$.

The author is grateful to Claude Carlet, Oktay Kasim-Zadeh and Maria Fedorova for helpful discussions.

References

1. P. Camion, C. Carlet, P. Charpin, N. Sendrier, On correlation-immune functions, Advances in Cryptology: Crypto '91, Proceedings, Lecture Notes in Computer Science, V. 576, 1991, pp. 86–100.
2. Seongtaek Chee, Sangjin Lee, Daiki Lee and Soo Hak Sung, On the Correlation Immune Functions and their Nonlinearity, Advances in Cryptology — Asiacrypt '96, Lecture Notes in Computer Science, V. 1163, 1996, pp. 232–243.
3. B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, R. Smolensky, The bit extraction problem or t-resilient functions, IEEE Symposium on Foundations of Computer Science, V. 26, 1985, pp. 396–407.
4. T. W. Cusick, On constructing balanced correlation immune functions, in Sequences and Their Applications, Proceedings of SETA '98, Springer Discrete Mathematics and Theoretical Computer Science, 1999, pp. 184-190.
5. M. Fedorova, Yu. Tarannikov. On the constructing of highly nonlinear resilient Boolean functions by means of special matrices, Cryptology ePrint archive (http://eprint.iacr.org/), Report 2001/083, October 2001, 16 pp.
6. M. Fedorova, Yu. Tarannikov. On the constructing of highly nonlinear resilient Boolean functions by means of special matrices, Progress in Cryptology — Indocrypt 2001, Chennai, India, December 16–20, 2001, Proceedings, Lecture Notes in Computer Science, V. 2247, pp. 254–266, Springer-Verlag, 2001.
7. E. Filiol, C. Fontaine, Highly Nonlinear Balanced Boolean Functions with a Good Correlation Immunity, Advanced in Cryptology, Eurocrypt '98, Helsinki, Finland, Lecture Notes in Computer Sciences, Vol. 1403, 1998, pp. 475–488.
8. S. Maitra, P. Sarkar, Highly nonlinear resilient functions optimizing Siegenthaler's Inequality, Crypto '99, Lecture Notes in Computer Science, Vol. 1666, 1999, pp. 198–215.
9. P. Sarkar, S. Maitra, Construction of nonlinear Boolean functions with important cryptographic properties, In Advanced in Cryptology: Eurocrypt 2000, Lecture Notes in Computer Science, V. 1807, 2000, pp. 485–506.
10. P. Sarkar, S. Maitra, Nonlinearity bounds and constructions of resilient Boolean functions, In Advanced in Cryptology: Crypto 2000, Proceedings, Lecture Notes in Computer Science, V. 1880, 2000, pp. 515–532.
11. E. Pasalic, S. Maitra, T. Johansson, P. Sarkar, New constructions of resilient and correlation immune Boolean functions achieving upper bounds on nonlinearity, WCC2001 International Workshop on Coding and Cryptography, Paris, January 8–12, 2001, Electronic Notes in Discrete Mathematics, Volume 6, Elsevier Science, 2001.

12. E. Pasalic, T. Johansson, Further results on the relation between nonlinearity and resiliency for Boolean functions, IMA Conference on Cryptography and Coding, Lecture Notes in Computer Science, Vol. 1746, 1999, pp. 35–44.
13. N. J. Patterson, D. H. Wiedemann, The covering radius of the $\left[2^{15}, 16\right]$ Reed–Muller code is at least 16276, IEEE Transactions on Information Theory, V. 29, No. 3, pp. 354–356, May 1983.
14. N. J. Patterson, D. H. Wiedemann, Correction to [13], IEEE Transactions on Information Theory, V. 36, No. 2, p. 443, March 1990.
15. O. S. Rothaus, On bent functions, Journal of Combinatorial Theory, Series A20, pp. 300–305.
16. J. Seberry, X. Zhang, Y. Zheng, On Constructions and Nonlinearity of Correlation Immune Functions, Advances in Cryptology, Eurocrypt '93, Proceedings, Lecture Notes in Computer Science, V. 765, 1993, pp. 181–199.
17. T. Siegenthaler, Correlation-immunity of nonlinear combining functions for cryptographic applications, IEEE Transactions on Information theory, V. IT-30, No 5, 1984, p. 776–780.
18. T. Siegenthaler, Decrypting a Class of Stream Ciphers Using Ciphertext Only, IEEE Transactions on Computer, V. C-34, No 1, Jan. 1985, pp. 81–85.
19. Yu. Tarannikov. On resilient Boolean functions with maximal possible nonlinearity, Cryptology ePrint archive (http://eprint.iacr.org/), Report 2000/005, March 2000, 18 pp.
20. Yu. Tarannikov. On resilient Boolean functions with maximal possible nonlinearity, Proceedings of Indocrypt 2000, Lecture Notes in Computer Science, V. 1977, pp. 19-30, Springer-Verlag, 2000.
21. Yu. Tarannikov. New constructions of resilient Boolean functions with maximal nonlinearity, 8th Fast Software Encryption Workshop, Preproceedings, Yokohama, Japan, April 2–4, 2001, pp. 70–81.
22. Yu. Tarannikov, P. Korolev, A. Botev. Autocorrelation coefficients and correlation immunity of Boolean functions, Proceedings of Asiacrypt 2001, Gold Coast, Australia, December 9–13, 2001, Lecture Notes in Computer Science, V. 2248, pp. 460–479, Springer-Verlag, 2001.
23. Y. Zheng, X. M. Zhang, Improved upper bound on the nonlinearity of high order correlation immune functions, Selected Areas in Cryptography, 7th Annual International Workshop, SAC2000, Lecture Notes in Computer Science, V. 2012, pp. 264–274, Springer-Verlag, 2001.

Optimized Self-Synchronizing Mode of Operation

Ammar Alkassar[1], Alexander Geraldy[2], Birgit Pfitzmann[2], and Ahmad-Reza Sadeghi[2]

[1] Sirrix AG, D-66424 Homburg, Germany. alkassar@sirrix.com
[2] Universität des Saarlandes, FR 6.2 Informatik, D-66123 Saarbrücken, Germany.
ageraldy@krypt.cs.uni-sb.de, {pfitzmann, sadeghi}@cs.uni-sb.de

Abstract. Modes of operation adapt block ciphers to many applications. Among the encryption modes, only CFB (Cipher Feedback) has both of the following properties: Firstly it allows transmission units shorter than the block-cipher length to be encrypted and sent without delay and message expansion. Secondly, it can resynchronize after the loss of such transmission units.
However, CFB is inefficient in such applications, since for every transmission unit, regardless how short, a call to the block cipher is needed.
We propose a new mode of operation based on CFB which remedies this problem. Our proposal, OCFB, is almost optimally efficient (i.e., almost as many message bits are encrypted as block-cipher output bits produced) and it can self-synchronize after the loss or insertion of transmission units. We prove the security of CFB and OCFB in the sense of modern cryptography.

1 Introduction

Symmetric-key block ciphers are one of the most prominent building blocks in many cryptographic systems. They are used to construct various primitives such as stream ciphers, message authentication codes, and hash functions. Conceptually, a block cipher is a function that maps fixed-size l-bit plaintext blocks to l-bit ciphertext blocks (l is called the length of block cipher). The function is parametrized by a key k of a certain length. Examples of well-known block ciphers are Triple-DES (based on [DES77]), IDEA [LM91] and the AES candidates [NIST00], in particular Rijndael [DR99]. To encrypt longer messages and to fulfil varying application requirements, several modes of operation for block ciphers have been proposed. Among the standardized encryption modes from [FIPS81], CBC (Cipher Block Chaining) and CFB (Cipher Feedback) use the previous ciphertext block in the encryption so that each ciphertext block depends on the preceding plaintext, while OFB (Output Feedback) acts as pseudo-random bit generator and allows a large part of the encryption procedure to be precomputed. CFB and OFB can be used for applications where plaintext units with $L < l$ bits ($L = 8$ and $L = 1$ are typical cases) must be encrypted and

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 78–91, 2002.

transmitted without delay. We call this a *transmission unit*. The counter mode (e.g., [DH79,LRW00]) can replace OFB.

Several applications need a *self-synchronizing* mode of operation, i.e., an error in the ciphertext must only lead to a small amount of incorrect plaintext. These are applications where the protocols have no or very basic built-in fault tolerance because the data type, e.g., voice or video, is such that small errors are unnoticeable or recoverable by natural redundancy. A more systematic alternative to a self-synchronizing cipher would be to add more error correction either to the transmission system or the application, but in practice, both may be fixed and one has to offer a transparent encryption layer in between. An application where we encountered this problem is ISDN, a common network for the subscriber area in Europe and Japan [Boc92], in particular for integrating phone, fax and Internet access. An effective way to secure the voice and fax communication, too, is transparent encryption directly before the ISDN network termination, e.g., by an ISDN card together with software encryption. Here one has precisely the network and application conditions for a self-synchronizing cipher: no underlying error correction, no message expansion possible (at least not without administrational problems), and the applications tolerate small errors but not desynchronization.

There are different types of errors. We speak of *bit errors* if certain bits are flipped, but the number of bits is unchanged. (Hence we include some burst errors in this class.) *Slips* are errors where bits are lost or inserted. Apart from outside disturbances, slips result from crossing networks with different clock frequency. CBC and CFB only tolerate slips if entire transmission units are inserted or lost. Hence only CFB can be used on networks where slips for units smaller than the block-cipher length occur. For instance, ISDN is byte-synchronous, i.e., slips can only be multiples of $L = 8$ bits [Boc92]. Hence CFB with 8-bit transmission units can be used. If nothing at all is known about the slips, $L = 1$ must be used. However, CFB is inefficient in such cases, since for every transmission unit the block cipher is called to encrypt l bits, e.g., 64 or 128. This is $n = l/L$ times as often as in other modes.

In this paper we present a solution OCFB to this problem. It is based on CFB, almost as efficient as CFB with $L = l$ and self-synchronizing even for $L < l$. We prove the concrete security of CFB and OCFB in the sense of [BDJR97]. To our knowledge this is also the first rigorous security analysis of CFB.

These properties make our proposal very appealing also to all applications which already use CFB.

Related literature: CFB and OCFB are so-called single modes of operation. Recently, multiple modes of operation have found considerable attention, but results as in [Bih94,Bih98,Wag98,HP99] suggest that single modes with a better block cipher are more promising. Now that an AES winner has been announced, this seems realistic.

Apart from that, recent research on modes of operation concentrates on MAC modes, e.g., [BR00,PR00,CKM00] and modes combining integrity and secrecy [GD00,Jut00,Rog00]. This is clearly important for many applications

and the natural way to build new applications, but as motivated above, well-established networks and applications remain where self-synchronization is important.

Proving the security of modes of operation started with [BKR94,BGR95], and a large part of the new literature cited above contains such proofs. Encryption modes (CBC and counter) were first treated in [BDJR97], and here also the concrete security of symmetric encryption was defined in detail.

The only fairly recent security analysis for CFB we know is [PNRB94], but rather from the point of view that CFB does not prevent differential and linear cryptanalysis of the block cipher.

Outline of the paper: We first present some basics and notations: the transmission model, notation for encryption, and CFB and its self-synchronization properties. After this we present our proposed operation mode OCFB and its self-synchronization property and compare the efficiency of CFB and OCFB. Finally, we present left-or-right security [BDJR97] as a basis for our security treatment and give concrete security proofs for both standard CFB and OCFB.

2 Preliminaries

2.1 Transmission Model

As motivated above, we assume a communication system with L-bit transmission units. The most important fact is that slips, i.e., losses or insertions, only occur for entire transmission units. The main case we consider is $L = 8$. We assume that we are not allowed any message expansion.

2.2 Encryption Notation

A block cipher is a triple (gen, enc, dec). The algorithm gen randomly generates a key k, which is used by the encryption function $enc_k : \{0,1\}^l \to \{0,1\}^l$ to encrypt a plaintext message m to a ciphertext $c = enc_k(m)$. The decryption function dec_k decrypts a ciphertext c' to $m' = dec_k(c')$ using the same key k. The value l is called the block-cipher length.[1]

CFB and OCFB use l-bit shift registers consisting of $n = l/L$ positions $SR[1], \ldots, SR[n]$ of L-bit transmission units. By $r = SR$ or $SR = r$ we denote reading or writing the entire register. Shifting a transmission unit m into the register is written $SR \leftarrow m$, i.e., this means $SR[i] = SR[i+1]$ for $i = 1, \ldots, n-1$ and $SR[n] = m$. Shifting a transmission unit out is written $m \leftarrow SR$, i.e., this means $m = SR[1]$, $SR[i] = SR[i+1]$ for $i = 1, \ldots, n-1$ and $SR[n] = 0$.

Other notations are $|m|$ for the length of a string and $\oplus$ for the bitwise exor of strings of equal length. By $a \in_R S$ we denote the random and uniform selection of an element a from the set S. Further, $P(pred(x) :: x = A(z))$ represents the probability that x fulfils a certain predicate $pred$ if x is chosen with the probabilistic algorithm A on input z.

[1] CFB and OCFB do not use dec_k or the fact that enc_k is a permutation; hence we could also define them with an arbitrary family F of functions enc_k.

2.3 Cipher Feedback Mode

The *Cipher Feedback Mode* (CFB) is illustrated in Figure 1. More precisely, $CFB^{l,L}$ denotes the version with an l-bit block cipher and L-bit transmission units, where $l = Ln$.

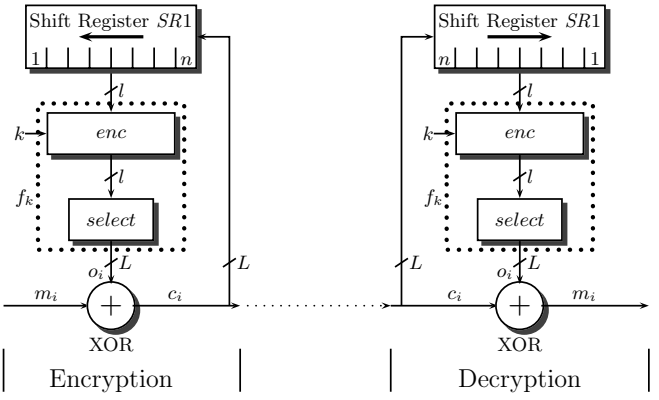


Fig. 1. Cipher Feedback Mode

In the figure, the block cipher enc_k followed by a fixed selection of L output bits is summarized as a function $f_k : \{0,1\}^l \to \{0,1\}^L$. The values m_i and c_i denote plaintext and ciphertext transmission units. A shift register $SR1$ holds the last n ciphertext transmission units, i.e., $SR1_i = c_{i-n} \dots c_{i-1}$ in Round $i > n$. In round $i = 1$, it contains an initialization vector $SR1_1 = IV$. Then the encryption algorithm of $CFB^{l,L}$ for the i-th transmission unit is:

1. $o_i = select(enc_k(SR1_i))$;
2. $c_i = m_i \oplus o_i$;
3. $SR1_{i+1} = (SR1_i \leftarrow c_i)$.

For decryption, Step 2 is replaced by $m_i = c_i \oplus o_i$.

Error Propagation: Under the given error model, i.e., if slips only occur for multiples of transmission units, errors disturb decryption only as long as they remain in the shift register of the receiver.

3 Optimized Cipher Feedback (OCFB)

For communication systems with $L \ll l$, CFB is inefficient. For instance, for $L = 8$, it calls enc_k for every single transmitted byte. This is less efficient than other modes by a factor of $n = l/L$, e.g., 8 for DES and 16 for AES.

The efficiency of CFB can be optimized by buffering all l output bits of enc_k and using them for successive transmission units, using a counter to trigger a

call of enc_k every n-th transmission unit. However, this would destroy the self-synchronization for slips of individual transmission units because the counters of the sender and the recipient would lose their synchrony (relative to the ciphertext stream). Hence the counters must be resynchronized. As the transmission model does not allow message expansion, this can only be done via the ciphertext itself. The idea is to use a *synchronization pattern*; this is sketched in Figure 2. Each automaton (for encryption and decryption) compares the content of its shift register $SR1$ with this pattern after each transmission unit. If it finds the pattern, it resets its counter; this reset synchronizes the counters of the two automata.

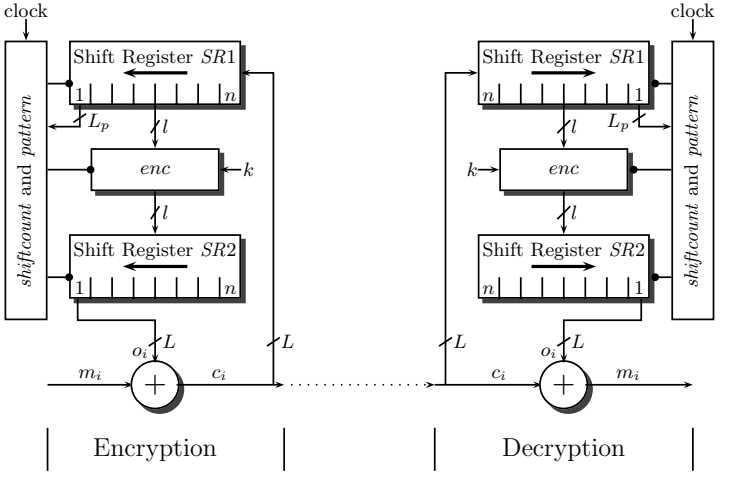


Fig. 2. OCFB

Let *pattern* denote the pattern, L_P its length, and p the probability that it matches a random register content. The pattern can be fixed at an arbitrary time, it need not be secret or random. The counter is called *shiftcount*, and *match* denotes the pattern-matching algorithm, typically a simple string comparison, possibly with wildcards. One can start with fixed initialization vectors in the registers $SR1$ and $shiftcount = n - 1$, but also asynchronously. Then the encryption algorithm of $OCFB^{l,L,p}$ for the i-th transmission unit is:

1. If $match(SR1_i, pattern)$ then $shiftcount = n$ else $shiftcount{+}{+}$;
2. if $shiftcount = n$ then $SR2_i = enc_k(SR1_i)$; $shiftcount = 0$;
3. $SR2_{i+1} = (o_i \leftarrow SR2_i)$;[2]
4. $c_i = m_i \oplus o_i$;
5. $SR1_{i+1} = (SR1_i \leftarrow c_i)$.

[2] If encryption is called in the next round, two values are called $SR2_{i+1}$. When in doubt, the second (final) one is meant.

For decryption, Step 4 is replaced by $m_i = c_i \oplus o_i$.

CFB can be seen as a special case of OCFB where the synchronization pattern is found in every round (e.g., by using the pattern length zero).

4 Efficiency of CFB and OCFB

We define the efficiency eff_M of a mode of operation M as the average number of encrypted plaintext bits per call to the l-bit block cipher, divided by l. The optimum with any normal mode of operation is therefore 1.The efficiency of $CFB^{l,L}$ is $\mathit{eff}_{CFB^{l,L}} = L/l$. The efficiency of OCFB is $E(X)/n$, where X, called *distance*, is the random variable describing the number of transmission units between two calls to the block cipher, and $E(X)$ its expectation. Recall that the pattern matches random data with probability p.Let $\overline{p} = 1 - p$, and for the pattern length we assume $L_p \leq L$.[3,4]

If $\mathit{shiftcount} = n$, the block cipher is always called. Hence, assuming that the values c_i are random (this will be justified in Section 5), the distance X has a cut-off geometric distribution: $P(X = j) = \overline{p}^{j-1}p$ for $0 < j < n$ and $P(X = n) = \overline{p}^{n-1}$. Hence

$$E(X) = \sum_{j=1}^{n} jP(X = j) = \sum_{j=1}^{n-1} jp\overline{p}^{j-1} + n\overline{p}^{n-1}$$
$$= \frac{1 - \overline{p}^n}{1 - \overline{p}}.$$

This can be verified by multiplying the terms out; a similar formula can, e.g., be found in [Nel95].Hence the efficiency is $\mathit{eff}_{OCFB^{l,L,p}} = (1 - \overline{p}^n)/(np)$.
For instance, for $l = 64$ and $L = 8$, an 8-bit pattern with $p = 2^{-8}$ gives an efficiency of 0.986, i.e., one block-cipher call for 7.89 bytes on average.

Self-synchronization for bit errors is as in CFB. Self-synchronization after a slip takes somewhat longer, $1/p$ transmission units on average. Tests have shown that this is no problem for phone and fax with the parameters from the previous example and the usual error rates of ISDN. Even with much larger p, e.g., $p = 1/n$ so that resynchronization after slips happens after about one l-bit block just as for bit errors, the efficiency of OCFB is still far superior to that of CFB. E.g., for $l = 64$ and $L = 8$, we then get $\mathit{eff}_{OCFB^{l,L,p}} = 1-(1-1/n)^n \approx 0.66$, which is more than 5 times faster than CFB.

[3] For our main example $L = 8$, this is reasonable and simplifies the analysis because successive matchings are independent. For small L (in particular, $L = 1$) we advise a pattern like 100...000 where repetition is only possible after L_p/L transmission units.

[4] Depending on the soft- or hardware configuration, it may be useful to buffer the values o_i to avoid problems if the pattern sometimes repeats faster than its expectation. In the definition, Step 4 becomes $c_i = m_i \oplus o_{i-\beta}$ for a buffer length β. The computation of the o_i's and the exors are then almost asynchronous. The security analysis is easily adapted to this case.

5 Security

In this section we prove the concrete security of CFB and OCFB, using the first proof as a basis for the second one. The proof follows the pattern of other security proofs of modes of operation: The block cipher is modeled as a pseudo-random function [GGM86] or rather, for concrete security, one parametrizes the effort an adversary needs to notice non-random properties [BKR94]. One then shows that any attack against the mode of operation that succeeds with a certain probability given a certain amount of resources would give an attack on the underlying block cipher, again with precise resources and probability.

5.1 Left-or-Right Security

Left-or-right security was introduced in [BDJR97] as a strong (adaptive) form of chosen-plaintext security.[5] The attack is modeled as a game between an active adversary (*left-right distinguisher*) D_{lr} and an encryption oracle $\mathcal{E}_{k,b}$, which contains a key k and a bit $b \in \{0,1\}$. In each round, D_{lr} chooses two plaintexts m_i^0 and m_i^1 with $|m_i^0| = |m_i^1|$ and gives them to $\mathcal{E}_{k,b}$. This oracle returns $c_i = enc_k(m_i^b)$. (The cases $b = 0$ and $b = 1$ are called left and right case.) Finally, D_{lr} outputs a bit e, meant as a guess at b. The adversary's advantage $Adv_{D_{lr}}$ is the defined as for a statistical test, i.e., the probability difference of the output $e = 0$ in the two cases.

The adversary's resources are parametrized by its maximum running time t, the number of queries q to the encryption oracle, and their total length μ. Its maximum success probability is called ε.

Definition 1 (Left-or-right security [BDJR97]). *An encryption scheme (gen, enc, dec) is (t, q, μ, ε)-secure in the left-or-right sense if for any adversary D_{lr} which runs in time at most t and asks at most q queries, these totaling at most μ bits,*

$$Adv_{D_{lr}} = P[{D_{lr}}^{\mathcal{E}_{k,0}} = 0 :: k \leftarrow gen] - P[{D_{lr}}^{\mathcal{E}_{k,1}} = 0 :: k \leftarrow gen] \leq \varepsilon.$$

The first term describes the probability that D_{lr} outputs $e = 0$ when interacting with the oracle containing $b = 0$, and the second term the corresponding probability for an oracle with $b = 1$. The definition is illustrated for CFB in Figure 3.

For modes of operation, the messages m_i^b that the adversary can choose adaptively may be individual transmission units.

Chosen-ciphertext security is not required in [BDJR97], and it cannot be required in the strict sense for a self-synchronizing cipher: This would correspond to non-malleability, but the purpose of self-synchronization is precisely to make slightly distorted ciphertexts decrypt to related cleartexts.

[5] More precisely, [BDJR97] contains four definitions and relations among them. It is shown that concrete encryption systems are best proven with respect to left-or-right security, since this implies good reductions to the other definitions.

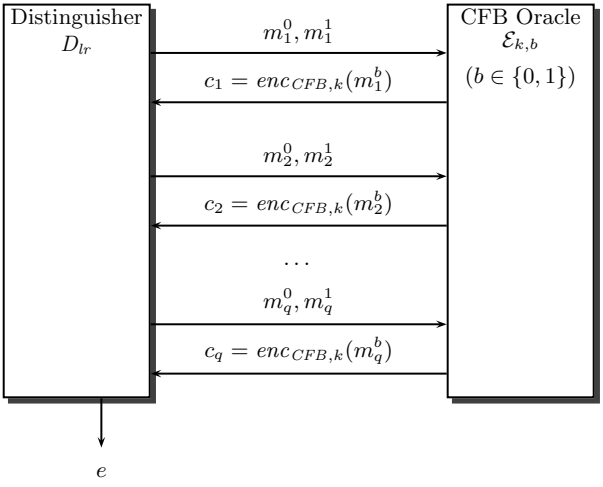


Fig. 3. Left-or-right security

5.2 Function Families

When considering a mode of operation, the block cipher is typically modeled as a pseudo-random function. In the case of CFB a weaker assumption is sufficient: Only the family of functions $f_k = select(enc_k(\cdot))$ must be pseudo-random (see Figure 1).

We use the concrete-security definitions from [BKR94,BDJR97]. An (l, λ)-*function family* is a multiset F of functions $f_k : \{0,1\}^l \to \{0,1\}^\lambda$. Alternatively, one writes $f \in_R F$ for the random choice of a function from the family, or $k \leftarrow gen$ if an algorithm is given that generates a key such that f_k is random in F.

The (l, λ)-*random function family* $R^{l,\lambda}$ consists of *all* functions $rf : \{0,1\}^l \to \{0,1\}^\lambda$. A key is simply an entire function rf. Clearly, such a key is much too long in practice, but pseudo-randomness of other (l, λ)-function families F is defined relative to $R^{l,\lambda}$: An adversary D_{prf} interacts with an oracle $\mathcal{F}$ that has chosen a function f randomly from either F or $R^{l,\lambda}$. In each round, D_{prf} may send a value $x \in \{0,1\}^l$ and $\mathcal{F}$ answers with $f(x)$. Again, the adversary has to output a bit meant as a guess at the function family, and its advantage is defined as for a statistical test.

Definition 2 (Pseudo-random functions [BDJR97]). *An (l, λ)-function family F is a (t, q, ε)-secure PRF family if for any distinguisher D_{prf} making at most q oracle queries and running in time at most t,*

$$Adv_{D_{prf}}(F) = P[{D_{prf}}^f = 0 :: f \in_R F] - P[{D_{prf}}^f = 0 :: f \in_R R^{l,\lambda}] \leq \varepsilon.$$

By $CFB^{l,L}(F)$ we denote CFB used with a certain function family F in the place of the functions $f_k = select(enc_k(\cdot))$. Similarly, $OCFB^{l,L,p}(F)$ denotes OCFB with F as the functions enc_k.

5.3 Security with Random Functions

Before proving a certain degree of concrete security, it is useful to consider what one can reasonably expect. This is a birthday bound, similar to other modes with feedback. The basic idea is that left-or-right security breaks down at the first repetition of the value of $SR1$: The adversary D_{lr} knows the values in $SR1$ (they are ciphertexts, at least after the initialization vector). If $SR1_i = SR1_j$ for $i \neq j$, then also $o_i = o_j$ in CFB. Hence $c_i \oplus c_j = m_i^b \oplus m_j^b$, and thus b is revealed if $m_i^0 \oplus m_j^0 \neq m_i^1 \oplus m_j^1$, which will typically be the case. For OCFB, a similar argument holds, but only those values in $SR1$ must be considered where encryption is called, i.e., on average they are fewer by a factor of $n \cdot \mathit{eff}_{OCFB}$.

We now show that if CFB were used with real random functions, there would indeed be no better attack than waiting for collisions among values of $SR1$, and we derive the resulting concrete security.

Lemma 1 (Concrete security of CFB with random functions). *CFB with random functions, i.e., $CFB^{l,L}(R^{l,L})$, is $(t, q, Lq, \varepsilon_{CFB^{l,L},q})$-secure in the left-or-right sense for any t (i.e., no runtime restriction), any number q of queries (of one transmission unit each), and $\varepsilon_{CFB^{l,L},q} = q(q-1)2^{-l-1}$.*

Proof. We abbreviate probabilities in the left-or-right game with bit b as P_b, and for intermediate values in such a game (e.g., c_i) we use the notation introduced in the text (see also Figure 3).For instance, the advantage can then be written $Adv_{D_{lr}} = P_0[e = 0] - P_1[e = 0]$.

We distinguish whether a collision occurs during the attack or not. Let C be the *collision event*, i.e., it contains all runs of the game where $i \neq j$ exist with $1 \leq i, j \leq q$ and $SR1_i = SR1_j$. Its complement is called $\overline{C}$.

As long as no collision has occurred, each value $o_i = f(SR1_i)$ can be seen as randomly and independently chosen (by the definition of random functions), i.e., it is a one-time pad for the plaintext transmission unit m_i^b. Hence c_i is random and independent of $c_1, \ldots, c_{i-1}$ and $m_1^b, \ldots, m_i^b$. Thus the collision probability in round i does not depend on b, and overall we can abbreviate

$$P[C] = P_0[C] = P_1[C]. \tag{1}$$

For the same reason, collisions are the only help for the adversary. If no collision occurs, the adversary outputs $e = 0$ with the same probability for $b = 0$ and $b = 1$.

$$P_0[e = 0 \mid \overline{C}] \;=\; P_1[e = 0 \mid \overline{C}]. \tag{2}$$

We can therefore rewrite the adversary's advantage as follows (using (1) and (2) in the last equality):

$$\begin{aligned} Adv_{D_{lr}} &= P_0[e=0] - P_1[e=0] \\ &= P_0[e=0 \mid C]\ P_0[C] + P_0[e=0 \mid \overline{C}]\ P_0[\overline{C}] \\ &\quad - P_1[e=0 \mid C]\ P_1[C] - P_1[e=0 \mid \overline{C}]\ P_1[\overline{C}] \\ &= P[C](P_0[e=0 \mid C] - P_1[e=0 \mid C]) \\ &\leq P[C]. \end{aligned}$$

Collisions: For the collision probability, we cannot simply use the birthday formula because $SR1_i$ and $SR1_j$ are not independent if $|j-i| < n$. We then say that $SR1_i$ and $SR1_j$ *overlap*.

We define the stream $B = IV\ c_1 \dots c_{q-1}$ of all the collision-relevant transmission units, i.e., those shifted through $SR1$ until the last, q-th, encryption. The length of B is $Q = (n+q-1)L$ bits, and the shift register contents are $SR1_i = B[i] \dots B[i+n-1]$ for $i = 1, \dots, q$. We first derive the number $col_{i,j}$ of streams with a collision $SR1_i = SR1_j$ for every possible pair (i,j), i.e., $1 \leq i < j \leq q$. We do this for the case with random initialization vector IV. (The difference in the other case is negligible).

a) Without overlapping, i.e., $j \geq i+n$:
   As $SR1_i = SR1_j$, there are 2^l possible values for the shift register contents of both rounds. The remaining $Q-2l$ bits offer 2^{Q-2l} possibilities. Thus $col_{i,j} = 2^{Q-l}$.
b) With overlapping, i.e., $i < j < i+n$:
   Let $d = j-i$. Then $SR1_i$ and $SR1_j$ together use $l+dL$ bits, and those have 2^{dL} possible values because $SR1_i[1] = SR1_i[1+d], \dots, SR1_i[n-d] = SR1_i[n]$. The remaining $Q-l-dL$ bits offer 2^{Q-l-dL} possibilities. Thus again $col_{i,j} = 2^{dL} \cdot 2^{Q-l-dL} = 2^{Q-l}$.

There are $q(q-1)/2$ possible pairs (i,j). Hence the number col of streams B with at least one collision is less than $q(q-1)2^{Q-l-1}$. (In the bound, streams with several collisions are counted several times.)

We have shown above that each stream *without* collision has the probability 2^{-Q} (because each new c_i is uniformly distributed). Hence $P[\overline{C}] = (2^Q - col)/2^Q > 1 - q(q-1)2^{-l-1}$. This implies

$$P[C] \leq q(q-1)2^{-l-1},$$

which remained to be shown. □

Lemma 2 (Concrete security of OCFB with random function).
For all parameters, OCFB with random functions is at least as secure as CFB for the same length of the block cipher and transmission units, i.e., $OCFB^{l,L,p}(R^{l,l})$ is at least as secure as $CFB^{l,L}(R^{l,L})$.

Proof. The structure of the proof is similar to that for CFB.

First we have to show again that each value o_i, and thus also c_i, is random and independent of $c_1, \ldots, c_{i-1}$ and $m_1^b, \ldots, m_i^b$. By the use of *shiftcount* in the encryption algorithm, $SR2$ is set to a value $rf(SR1_i)$ at least every n transmission units, where rf is the random function replacing enc_k. This gives n potential values $o_i, \ldots, o_{i+n-1}$ which are totally random at this point in time. The adversary may choose $m_{i+1}^b, \ldots, m_{i+n-1}^b$ adaptively after this, but no information at all about o_{i+j} is given out before its use in c_{i+j}. Thus the claimed independence holds. This implies, as in the proof of Lemma 1, that $Adv_{D_{lr}} \leq P[C_{enc}]$, where C_{enc} is the event that there is a collision $SR1_i = SR1_j$ for two indices i, j where encryption (i.e., here rf) is called.

Clearly $P[C_{enc}] < P[C]$, the probability of collisions for arbitrary indices i, j. Hence, with the results of the previous paragraph, an upper bound for $P[C]$ can be computed literally as for CFB. □

This lemma shows that $OCFB^{l,L,p}(R^{l,l})$ is $(t, q, Lq, \varepsilon_{CFB^{l,L},q})$-secure for all t and q. In fact, the security is even somewhat better: It is approximately $(t, q, Lq, \varepsilon_{OCFB^{l,L,p},q})$, where $\varepsilon_{OCFB^{l,L,p},q} = q'(q'-1)2^{-l-1}$ with $q' = q/(n \cdot \mathit{eff}_{OCFB^{l,L,p}})$. This follows by counting only the collisions in C_{enc} in the second part of the proof: On average, encryption is called for every $(n \cdot \mathit{eff}_{OCFB^{l,L,p}})$-th index. Thus q' roughly plays the role of q in the proof above, i.e., the number of possibilities for i and j.

5.4 Security with Pseudo-Random Functions

Now we want to prove the security of CFB and OCFB with pseudo-random functions, i.e., in the real world, as far as the block cipher is pseudo-random.

Theorem 1 (Concrete security of CFB with pseudo-random functions).
Let F be a (t', q', ε')-secure (l, L)-bit pseudo-random function family, and t_{CFB} the time needed for one CFB round without the computation of f_k. Then $CFB^{l,L}(F)$ is (t, q, μ, ε)-secure in the left-or-right sense with $q = q'$, $\mu = q'L$, $t = t' - qt_{CFB} - t_{const}$ for a small constant t_{const}, and $\varepsilon = 2\varepsilon' + \varepsilon_{CFB^{l,L},q}$.

These bounds are very good: The allowed time t for an attack is almost t' (because t' is the time for an attack on F, while qt_{CFB} is time needed for the correct use of CFB), and an addition of $2\varepsilon'$ is standard for this kind of proof (see [BDJR97]).

The basic idea of the proof is the standard pseudo-randomness argument: If an adversary could break left-or-right security of CFB with a pseudo-random function family better than this is possible with random functions, one could use this adversary as a distinguisher between real and pseudo-random functions. The following reduction gives the concrete security for this argument.

Proof. We assume that a distinguisher D_{lr} contradicts the theorem. We construct a distinguisher D_{prf} which contains D_{lr} as a black box, playing the left-or-right

game with a CFB oracle, see Figure 4. The CFB oracle is implemented by D_{prf} itself, using its own oracle $\mathcal{F}$ (containing a random or pseudo-random function) as f_k. Hence D_{prf} chooses b itself (randomly) and can compare D_{lr}'s output e with b. It outputs the Boolean value $e^* = \neg(e = b)$.

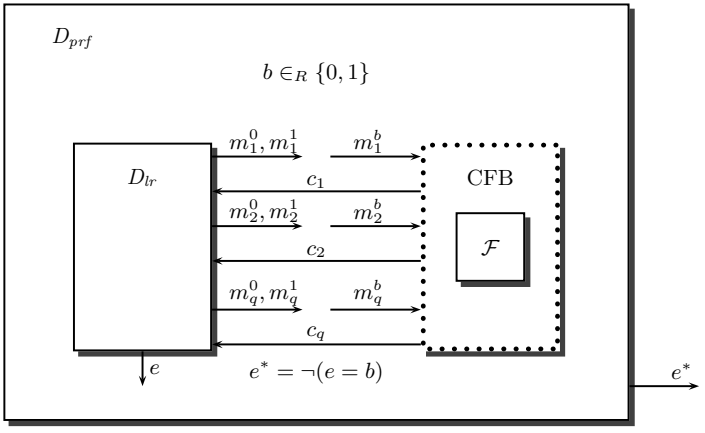


Fig. 4. Reduction of CFB to PRF

For each query of D_{lr} to the assumed CFB oracle, D_{prf} must make one query to $\mathcal{F}$ and update the CFB state. Hence it makes $q' = q$ queries and takes time $t' = t + qt_{CFB} + t_{const}$, where t_{const} is the time for set-up and the final comparison. Hence D_{prf} is a distinguisher against which the function family F is assumed to be secure.

We now abbreviate $R^{l,L}$ by R, and for $G \in \{R, F\}$ we write $Adv_{D_{lr}}(G)$ for the advantage of D_{lr} (according to Definition 2) when $CFB^{l,L}(G)$ is used. Furthermore, let P_F and P_R denote the two probability spaces from Definition 2. Then we can write the advantage of D_{prf} as

$$Adv_{D_{prf}} = P_F[e^* = 0] - P_R[e^* = 0],$$

and for $G \in \{R, F\}$, we can continue

$$\begin{aligned} P_G[e^* = 0] &= P[e^* = 0 :: f \in_R G; e^* \leftarrow {D_{prf}}^f] \\ &= P[e = b :: f \in_R G; b \in_R \{0,1\}; e \leftarrow {D_{lr}}^{\mathcal{CFB}_{f,b}}]. \end{aligned}$$

Here we used the construction of D_{prf} and introduced the notation $\mathcal{CFB}_{f,b}$ (corresponding to Definition 1) for a CFB left-or-right oracle with the particular function f and bit b. As the choices of f and b are independent, b can be chosen first and we get

$$P_G[e^* = 0] = \frac{1}{2} \sum_{b \in \{0,1\}} P[D_{lr}{}^{\mathcal{CFB}_{f,b}} = b :: f \in_R G]$$
$$= \frac{1}{2}(P[D_{lr}{}^{\mathcal{CFB}_{f,0}} = 0 :: f \in_R G] + 1 - P[D_{lr}{}^{\mathcal{CFB}_{f,1}} = 0 :: f \in_R G])$$
$$= \frac{1}{2} + \frac{1}{2} Adv_{D_{lr}}(G).$$

We have assumed $Adv_{D_{lr}}(F) > \varepsilon$, and by Lemma 1, $Adv_{D_{lr}}(R) \leq \varepsilon_{CFB^{l,L},q}$. Hence

$$Adv_{D_{prf}} = \frac{1}{2}(Adv_{D_{lr}}(F) - Adv_{D_{lr}}(R)) > \frac{1}{2}(\varepsilon - \varepsilon_{CFB^{l,L},q}) = \varepsilon'.$$

This is the desired contradiction to the security of the pseudo-random function family F. □

Theorem 2 (Concrete security of OCFB with pseudo-random functions).
OCFB is as secure as CFB in the following sense: Let F be a (t', q', ε')-secure (l, l)-bit pseudo-random function family. Then $OCFB^{l,L,p}(F)$ is (t, q, μ, ε)-secure in the left-or-right sense with $q = q'$, $\mu = qL$, $t = t' - qt_{OCFB} - t_{const}$, and $\varepsilon = 2\varepsilon' + \varepsilon_{CFB^{l,L},q}$.

Proof. This proof is almost identical to that for CFB. The only difference is that D_{lr} simulates an OCFB-oracle instead of a CFB-oracle, and therefore the time for each call is t_{OCFB} instead of t_{CFB}. □

In fact, the security is better: On average, D_{lr} only needs $q' = q/(n \cdot \mathit{eff}_{OCFB^{l,L,p}})$ calls to its own oracle $\mathcal{F}$ (corresponding to the block cipher) to encrypt the q transmission units, i.e., q can be chosen correspondingly larger.

Furthermore, as discussed after Lemma 2, the bound $\varepsilon_{CFB^{l,L},q}$ in that lemma is not quite tight. The proof of Theorem 2 remains unchanged for whatever value $\varepsilon_{OCFB^{l,L,p},q}$ is proven in Lemma 1.

Acknowledgment. We thank Michael Braun, Jörg Friedrich, Matthias Schunter, and Michael Waidner for interesting discussions and helpful comments.

References

[BDJR97] M. Bellare, A. Desai, E. Jokipii, P. Rogaway: A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation, 38th Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1997, 394–403.

[BGR95] M. Bellare, R. Guérin, P. Rogaway: XOR MACs: New Methods for Message Authentication Using Finite Pseudorandom Functions; Crypto '95, LNCS 963, Springer-Verlag, Berlin 1995, 15–28.

[Bih94] E. Biham: On Modes of Operation; Fast Software Encryption '93, LNCS 809, Springer-Verlag, Berlin 1994, 116–120.

[Bih98] E. Biham: Cryptanalysis of Multiple Modes of Operation; Journal of Cryptography 11/1 (1998) 45–58.

[BKR94] M. Bellare, J. Kilian, P. Rogaway: The security of cipher block chaining; Crypto '94, LNCS 839, Springer-Verlag, Berlin 1994, 341–358.

[Boc92] P. Bocker: ISDN – The Integrated Services Digital Network; (2nd ed.), Springer-Verlag, Berlin 1992.

[BR00] J. Black, P. Rogaway: CBC macs for arbitrary-length messages: the three-key constructions; Crypto 2000, LNCS 1880, Springer-Verlag, Berlin 2000, 197–215.

[CKM00] D. Coppersmith, L. R. Knudsen, C. J. Mitchell: Key recovery and forgery attacks on the MacDES MAC algorithm; Crypto 2000, LNCS 1880, Springer-Verlag, Berlin 2000, 184–196.

[DES77] Specification for the Data Encryption Standard; Federal Information Processing Standards Publication 46 (FIPS PUB 46), 1977.

[DH79] W. Diffie, M. E. Hellman: Privacy and Authentication: An Introduction to Cryptography; Proceedings of the IEEE 67/3 (1979) 397–427.

[DR99] J. Daemen, V. Rijmen: The Rijndael Block Cipher, AES Proposal; `http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf,1999`.

[FIPS81] DES Modes of Operation; Federal Information Processing Standards Publication 81 (FIPS PUB 81) December 2, 1980.

[GD00] V. Gligor, P. Donescu: Integrity-Aware PCBC Encryption Schemes; Security Protocols 1999, LNCS 1796, Springer-Verlag, Berlin 2000, 153–171.

[GGM86] O. Goldreich, S. Goldwasser, S. Micali: How to construct random functions. Journal of the ACM 33/4 (1986) 210–217.

[HP99] H. Handschuh, B. Preneel: On the Security of Double and 2-Key Triple Modes of Operation; 6th International Workshop on Fast Software Encryption, LNCS 1636, Springer-Verlag, Berlin 1999, 215–230.

[Jut00] C. S. Jutla: Encryption Modes with Almost Free Message Integrity; NIST Workshop Symmetric Key Block Cipher Modes of Operation, Baltimore, October 2000, `http://csrc.nist.gov/encryption/aes/modes/`.

[LM91] X. Lai, J. Massey: A Proposal for a New Block Encryption Standard; Eurocrypt '90, LNCS 473, Springer-Verlag, Berlin 1991, 389–404.

[LRW00] H. Lipmaa, P. Rogaway, D. Wagner: CTR-Mode Encryption; NIST Workshop Symmetric Key Block Cipher Modes of Operation, Baltimore, October 2000, `http://csrc.nist.gov/encryption/aes/modes/`.

[Nel95] Randolph Nelson: Probability, Stochastic Processes, and Queuing Theory, Springer 95

[NIST00] National Institute of Standards (NIST): AES — Advanced Encryption Standard (AES) Development Effort; `http://csrc.nist.gov/encryption/aes/`, 1997–2000.

[PNRB94] B. Preneel, M. Nuttin, V. Rijmen, J. Buelens: Cryptanalysis of DES in the CFB mode; Crypto '93, LNCS 773, Springer-Verlag, Berlin 1994, 212–223.

[PR00] E. Petrank, C. Rackoff: CBC MAC for Real-Time Data Sources; Journal of Cryptology 13/3 (2000) 315–338.

[Rog00] P. Rogaway: OCB Mode: Parallelizable Authenticated Encryption; NIST Workshop Symmetric Key Block Cipher Modes of Operation, Baltimore, October 2000, `http://csrc.nist.gov/encryption/aes/modes/`.

[Wag98] D. Wagner: Cryptanalysis of some Recently-Proposed Multiple Modes of Operation; Fast Software Encryption '98, LNCS 1372, Springer-Verlag, Berlin 1998, 254–269.

Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes

Virgil D. Gligor* and Pompiliu Donescu

VDG Inc., 6009 Brookside Drive, Chevy Chase, MD 20815
{gligor,pompiliu}@eng.umd.edu

Abstract. We present the eXtended Ciphertext Block Chaining (XCBC) and the eXtended Electronic Codebook (XECB) encryption schemes or modes of encryption that can detect encrypted-message forgeries with high probability even when used with typical non-cryptographic Manipulation Detection Code (MDC) functions (e.g., bit-wise exclusive-or and cyclic redundancy code (CRC) functions). These modes detect encrypted-message forgeries at low cost in performance, power, and implementation, and preserve both message secrecy and integrity in a single pass over the message data. Their performance and security scale directly with those of the underlying block cipher function. We also present the XECB message authentication (XECB-MAC) modes that have all the operational properties of the XOR-MAC modes (e.g., fully parallel and pipelined operation, incremental updates, and out-of-order verification), and have better performance. They are intended for use either stand-alone or with encryption modes that have similar properties (e.g., counter-based XOR encryption). However, the XECB-MAC modes have higher upper bounds on the probability of adversary's success in producing a forgery than the XOR-MAC modes.

1 Introduction

No one said this was an easy game !
Paul van Oorschot, March 1999.

A long-standing goal in the design of block encryption modes has been the ability to provide message-integrity protection with simple Manipulation Detection Code (MDC) functions, such as the exclusive-or, cyclic redundancy code (CRC), or even constant functions [5,7,9]. Most attempts to achieve this goal in the face of chosen-plaintext attacks focused on different variations of the Cipher Block Chaining (CBC) mode of encryption, which is the most common block-encryption mode in use. To date, most attempts, including one of our own, failed [8].

* This work was performed while this author was on sabbatical leave from the University of Maryland, Department of Electrical and Computer Engineering, College Park, Maryland 20742.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 92–108, 2002.

In this paper, we define the eXtended Ciphertext Block Chaining (XCBC) modes and the eXtended Electronic Codebook (XECB) encryption modes that can be used with an exclusive-or function to provide the authentication of encrypted messages in a single pass over the data with a single cryptographic primitive (i.e., the block cipher). These modes detect integrity violations at a low cost in performance, power, and implementation, and can be executed in a parallel or pipelined manner. They provide authentication of encrypted messages in real-time, without the need for an additional processing path over the input data. The performance and security of these modes scale directly with the performance and security of the underlying block cipher function since separate cryptographic primitives, such as hash functions, are unnecessary.

We also present the XECB message authentication (i.e., XECB-MAC) modes and their salient properties. The XECB-MAC modes have all the operational properties of the XOR message authentication (XOR-MAC) modes (e.g., they can operate in a fully parallel and pipelined manner, and support incremental updates and out-of-order verification [2]), and have better performance; i.e., they use only about half the number of block-cipher invocations required by the XOR-MAC modes. However, the XECB-MAC modes have higher bounds on the adversary's success of producing a forgery than those of the XOR-MAC modes. The XECB-MAC modes are intended for use either stand-alone to protect the integrity of plaintext messages, or with encryption modes that have similar properties (e.g., counter-based XOR encryption [1] a.k.a "counter mode") whenever it is desired that separate keys be used for secrecy and integrity modes.

2 An Integrity Mode for Encryption

Preliminaries and Notation. In defining the encryption modes we adopt the approach of Bellare *et al.* (viz., [1]), who show that an encryption mode can be viewed as the triple (E, D, KG), where E is the encryption function, D is the decryption function, and KG is the probabilistic key-generation algorithm. (Similarly, a message authentication (MAC) mode can be viewed as the triple (S, V, KG), where S is the message signing function, V is the message verification function, and KG is the probabilistic key-generation algorithm.) Our encryption and authentication modes are implemented with block ciphers, which are modeled with finite families of pseudorandom functions (PRFs) or pseudorandom permutations (PRPs).

In this context, we use the concepts of pseudorandom functions (PRFs), pseudorandom permutations (PRPs), and super-pseudorandom permutations (SPRPs) ([1], [15]). Let $R^{l,L}$ the set of all functions $\{0,1\}^l \to \{0,1\}^L$. We use F to denote either a family of pseudorandom functions or a family of super-pseudorandom permutations, as appropriate (e.g., for the encryption schemes, F will be a family of super-pseudorandom permutations, while for our MAC schemes, F can be a family of pseudorandom functions).

Given encryption scheme $\Pi = (E, D, KG)$ that is implemented with SPRP F, we denote the use of the key $K \xleftarrow{R} KG$ in the encryption of a plaintext string

x by $E^{F_K}(x)$, and in the decryption of ciphertext string y by $D^{F_K}(y)$. The most common method used to detect modifications of encrypted messages applies a MDC function g (e.g., a non-keyed hash, cyclic redundancy code (CRC), bitwise exclusive-or function [16]) to a plaintext message and concatenates the result with the plaintext before encryption with E^{F_K}. A message thus encrypted can be decrypted and accepted as valid only after the integrity check is passed; i.e., after decryption with D^{F_K}, the concatenated value of function g is removed from the plaintext, and the check passes only if this value matches that obtained by applying the MDC function to the remaining plaintext [5,7,16]. If the integrity check is not passed, a special failure indicator, denoted by $Null$ herein, is returned. This method[1] has been used in commercial systems such as Kerberos V5 [18,22] and DCE [6,22], among others. The encryption scheme obtained by using this method is denoted by Π-g = (E-g,D-g,KG), where Π is said to be *composed* with MDC function g. In this mode, we denote the use of the key K in the encryption of a plaintext string x by (E^{F_K}-g)(x), and in the decryption of ciphertext string y by (D^{F_K}-g)(y).

A design goal for Π-g = (E-g, D-g, KG) modes is to find the simplest encryption mode Π = (E,D,KG) (e.g., comparable to the CBC modes) such that, when this mode is composed with a simple, non-cryptographic MDC function g (e.g., as simple as a bitwise exclusive-or function), message encryption is protected against *existential forgeries*. For any key K, a forgery is any ciphertext message that is not the output of E^{F_K}-g. An existential forgery (EF) is a forgery that passes the integrity check of D^{F_K}-g upon decryption; i.e., for forgery y', (D^{F_K}-g)(y') $\neq Null$, where $Null$ is a failure indicator. Note that the plaintext outcome of an existential forgery need not be known to the forgerer. It is sufficient that the receiver of a forged ciphertext decrypt the forgery correctly.

Message Integrity Attack: Existential Forgery in a Chosen-Plaintext Attack. The attack is defined by a protocol between an adversary A and an oracle O[2] as follows.

1. A and O select encryption mode Π-g = (E-g,D-g,KG), and O selects, uniformly at random, a key K of KG.
2. A sends encryption queries (i.e., plaintext messages to be encrypted) x^p, $p = 1, \cdots, q_e$, to the encryption function of O. Oracle O responds to A by returning $y^p = (E^{F_K}\text{-}g)(x^p)$, $p = 1, \cdots, q_e$, where x^p are A's chosen plaintext messages. A records both its encryption queries and O's responses to them.
3. After receiving O's encryption responses, A forges a collection of ciphertexts $y'^i, 1 \leq i \leq q_v$ where $y'^i \neq y^p, \forall p = 1, \cdots, q_e$, and sends each decryption

[1] Note that other methods for protecting the integrity of encrypted messages exist; e.g., encrypting the message with a secret key and then taking the separately keyed MAC of the ciphertext [16,3]. These methods require two passes over the message data, require more power, and are more complex to implement than the modes we envision for most common use. Nevertheless these methods are useful whenever key separation is desired for secrecy and integrity.

[2] O can be viewed as two oracles, the first for the encryption function of O and the second for the decryption function of O.

query y'^i to the decryption function of O. O returns a success or failure indicator to A, depending on whether $(D^{F_K}\text{-}g)(y'^i) \neq Null$.

Adversary A is successful if there is at least one decryption query y'^i such that $(D^{F_K}\text{-}g)(y'^i) \neq Null$ for $1 \leq i \leq q_v$; i.e., y'^i is an existential forgery. The mode Π-g = (E-g,D-g,KG) is said to be secure against a message-integrity attack if the probability of an existential forgery in a chosen-plaintext attack is negligible. (We use the notion of negligible probability in the same sense as that of Naor and Reingold [17].)

Attack Parameters. A is allowed q_e encryption queries (i.e., queries to E^{F_K}-g), and q_v decryption queries (i.e., queries to D^{F_K}-g) totaling $\mu_e + \mu_v$ bits, and taking time $t_e + t_v$.

Parameters q_e, μ_e, t_e are bound by the parameters $(q', \mu', t', \epsilon')$ which define the chosen-plaintext security of Π = (E,D,KG) in a secrecy attack (e.g., in the left-or-right sense [1], for instance), and a constant c' determined by the speed of the function g. Since parameters $(q', \mu', t', \epsilon')$ are expressed in terms of the given parameters (t, q, ϵ) of the SPRP family F, the attack parameters can be related directly to those of the SPRP family F.

Parameters $q_e, \mu_e, t_e, q_v, \mu_v, t_v$ are also bound by the parameters (t, q, ϵ) of the SPRP family F, namely $\mu_e + \mu_v \leq ql$, and $t_e + t_v \leq t$. (The parameters q_e, q_v are determined by μ_e, μ_v.) These parameters can be set to specific values determined by the desired probability of adversary's success. Note that $q_v > 0$ since A must be allowed verification queries. Otherwise, A cannot test whether his forgeries are correct, since A does not know key K.

The message-integrity attack defined above is not weaker than an adaptive one in the sense that the success probability of adversary A bounds from above the success probability of another adversary A' that intersperses the q_e encryption and q_v verification queries; i.e., the adversary is allowed to make his choice of forgery after seeing the result of legitimate encryptions and other forgeries. (This has been shown for chosen-message attacks against MAC functions [2], but the same argument holds here.) To date, this is the strongest of the known goal-attack combinations against the integrity (authentication) of encrypted messages [3,10].

3 Definition of the XCBC and XCBC-XOR Modes

We present three XCBC modes, namely (1) stateless, (2) stateful-sender, and (3) stateful modes, and some implementation options. In general, the fewer state variables the more robust the mode is in the face of failures (or disconnections) and intrusion. This might suggest that, in practice, stateless modes are preferable. However, this may not always be the case because (1) the cost of invoking a good source of randomness for each message can be high (e.g., substantially higher than a single AES block encryption), (2) the random number used in each message encryption by the sender must be securely transmitted to the receiver, which usually costs an additional block-cipher invocation, and (3) the source of

randomness may be hard to protect. The stateful-sender mode (e.g., a counter-based mode) eliminates the need for a good source of randomness but does not always eliminate the extra block-cipher invocation and the need to protect the extra sender state variables; e.g., the source of randomness is replaced by the enciphering of a message counter but the counter must be maintained and its integrity must be protected by the sender across multiple message authentications. Any erroneous or faulty modification of the message counter must trigger re-keying. (The other advantage of counter-based modes, namely the ability to go beyond the "birthday barrier" when used with pseudo-random functions, does not materialize in the context of the AES since AES is modeled as a family of pseudo-random permutations.)

Maintaining secret shared-state variables, as opposed to just sender-state, helps eliminate the extra block-cipher invocations. Extending the shared keying state with extra, per-key, random variables shared by senders and receivers is a fairly straight-forward matter; e.g., these shared variables can be generated and distributed in the same way as the shared secret key, or can be generated using the shared key (at some marginal extra cost per message) by encrypting constants with the shared key. However, maintaining the shared state in the face of failures (or disconnections), and intrusion presents an extra challenge for the mode user; e.g., enlarging the shared state beyond that of a shared secret key may increase the exposure of the mode to physical attacks. The above discussion suggests that none of the three types of operational modes is superior to the others in all environments, and hence all of them should be supported in a general mode definition.

In the encryption modes presented below, the key generation algorithm, KG, outputs a random, uniformly distributed, k-bit string or key K for the underlying $SPRP$ family F, thereby specifying $f = F_K$ and $f^{-1} = F_K^{-1}$ of l-bits to l-bits. If a separate second key is needed in a mode, then a new string or key K' is generated by KG identifying $f' = F_{K'}$ and $f'^{-1} = F_{K'}^{-1}$. The plaintext message to be encrypted is partitioned into a sequence of l-bit blocks (padding is done first, if necessary), $x = x_1 \cdots x_n$. Throughout this paper, $\oplus$ is the *exclusive-or* operator and $+$ represents *modulo* 2^l *addition.*

Stateless XCBC Mode (XCBC$)
The encryption and decryption functions of the stateless mode, $\mathcal{E}{-}XCBC\$^{F_K}(x)$ and $\mathcal{D}{-}XCBC\$^{F_K}(y)$, are defined as follows.

function $\mathcal{E}$−XCBC$$^f(x)$
$r_0 \leftarrow \{0,1\}^l$
$y_0 = f(r_0);\ z_0 = f'(r_0)$
for $i = 1, \cdots, n$ **do** {
$z_i = f(x_i \oplus z_{i-1})$
$y_i = z_i + i \times r_0$ }
return $y = y_0 || y_1 y_2 \cdots y_n$

function $\mathcal{D}$−XCBC$$^f(y)$
Parse y as $y_0 || y_1 \cdots y_n$
$r_0 = f^{-1}(y_0);\ z_0 = f'(r_0)$
for $i = 1, \cdots, n$ **do** {
$z_i = y_i - i \times r_0$
$x_i = f^{-1}(z_i) \oplus z_{i-1}$ }
return $x = x_1 x_2 \cdots x_n$

Stateful-Sender XCBC Mode (XCBCC)

The encryption and decryption functions of the stateful-sender mode, $\mathcal{E}{-}XCBCC^{F_K}(x, ctr)$ and $\mathcal{D}{-}XCBCC^{F_K}(y)$, are defined as follows.

function $\mathcal{E}$−XCBCC$^f(x, ctr)$
$r_0 = f(ctr);\ z_0 = f'(r_0)$
for $i = 1, \cdots, n$ **do** {
$z_i = f(x_i \oplus z_{i-1})$
$y_i = z_i + i \times r_0$ }
$ctr' \leftarrow ctr + 1$
$y = ctr||y_1 y_2 \cdots y_n$
return y

function $\mathcal{D}$−XCBCC$^f(y)$
Parse y as $ctr||y_1 \cdots y_n$
$r_0 = f(ctr);\ z_0 = f'(r_0)$
for $i = 1, \cdots, n$ **do** {
$z_i = y_i - i \times r_0$
$x_i = f^{-1}(z_i) \oplus z_{i-1}$ }
return $x = x_1 x_2 \cdots x_n$

Note that in the XCBCC mode the counter ctr can be initialized to a known constant such as -1 by the sender. ctr' represents the updated ctr value. In both of the above modes the complexity is $n + 2$ block-cipher invocations, where n in the length of input string x in blocks.

Stateful XCBC Mode (XCBCS)

Let IV be a random and uniformly distributed variable that is part of the keying state shared by the sender and receiver.
$\mathcal{E}{-}XCBCS\$^{F_K}(x)$ and $\mathcal{D}{-}XCBCS\$^{F_K}(y)$, are defined as follows.

function $\mathcal{E}$−XCBCS$\$^f(x)$
$r_0 \leftarrow \{0,1\}^l$
$y_0 = f(r_0);\ z_0 = IV + r_0$
for $i = 1, \cdots, n$ **do** {
$z_i = f(x_i \oplus z_{i-1})$
$y_i = z_i + i \times r_0$ }
return $y = y_0||y_1 y_2 \cdots y_n$

function $\mathcal{D}$−XCBCS$\$^f(y)$
Parse y as $y_0||y_1 \cdots y_n$
$r_0 = f^{-1}(y_0);\ z_0 = IV + r_0$
for $i = 1, \cdots, n$ **do** {
$z_i = y_i - i \times r_0$
$x_i = f^{-1}(z_i) \oplus z_{i-1}$ }
return $x = x_1 x_2 \cdots x_n$

Note that in the XCBCS mode the shared IV value can be generated randomly by KG and distributed to the sender and receiver along with key K thereby saving one block cipher invocation, or can be can be generated using key K by standard key-separation techniques thereby requiring an additional block encryption operation per key. In the former case, the complexity of the mode is exactly $n+1$ block-cipher invocations and, in the latter, is *asymptotically* $n + 1$ block-cipher invocations.

Chaining Sequence. The *block chaining sequence* is that used for the traditional CBC mode, namely $z_i = f(x_i \oplus z_{i-1})$, where z_0 is the initialization vector, x_i is the plaintext and z_i is the ciphertext of block $i, i = 1, \cdots, n$. In contrast with the traditional CBC mode, the value of z_i is not revealed outside the encryption modes, and, for this reason, z_i is called a *hidden* ciphertext block. The actual ciphertext output, y_i, of the XCBC modes is defined using extra randomization, namely $y_i = z_i + i \times r_0$, where $i \times r_0$ is the *modulo* 2^l *addition* of the random, uniformly distributed, variable r_0, i times to itself; i.e., $i \times r_0 \stackrel{def}{=} \underbrace{r_0 + \cdots + r_0}_{i\ times}$.

Examples for why the randomization is necessary include those which show that, without randomization, the swapping of two z_i blocks of a ciphertext mes-

sage, or the insertion of two arbitrary but identical blocks into two adjacent positions of a ciphertext message, would cause the decryption of the resulting forgery with probability one whenever an bitwise exclusive-or function is used as the MDC (which is what we intend to use, since these functions are among the fastest known). Correct randomization sequences, such as $i \times r_0$, ensure that, among other things, collisions between any two z_i values is negligible regardless of whether these values are obtained during message encryption, forgery decryption, or both. Note that this probability is negligible even though the randomization sequence $i \times r_0$ allows low-order bits of some z_i's to become known. (A detailed account of why such collisions contribute to an adversary's success in breaking message integrity is provided in the proof of the XCBC$-XOR mode; viz., *http://csrc.nist.gov/encryption/modes/proposedmodes.*) Examples of incorrect randomization sequences can be readily found; e.g., the sequence whereby each element i is computed as an bitwise exclusive-or of i instances of r_0.

Initialization. In *stateless* implementations of the XCBC modes $r_0 \leftarrow \{0,1\}^l$; i.e., r_0 is initialized to a random, uniformly distributed, l-bit value for every message. The value of r_0 is sent by the sender to the receiver as $y_0 = f(r_0)$. In contrast, in *stateful-sender* implementations, which avoid the use of a random number generator, a counter, *ctr*, is initialized to a new l-bit constant (e.g., -1) for every key K, and incremented on every message encryption. In *stateful* implementations, a random initialization-vector value IV that is shared by the sender and receiver is generated for every key K, and used to create a per-message random initialization vector z_0.

In all XCBC modes, the initialization vector z_0 is independent of r_0. While non-independent z_0 and r_0 values might yield secure initialization, simple relationships between these values can lead to the discovery of r_0 with non-negligible probability, and integrity can be easily broken.[3] Since we use z_0 in the definition of function $g(x)$ (see below), z_0 should also be unpredictable so that $g(x)$ has a per-message unpredictable value.

The choice of encrypting r_0 with a second key K' to obtain z_0 (i.e., $z_0 = f'(r_0)$) is made exclusively to simplify both the secrecy [1] and the integrity proofs; e.g., such a z_0 is independent of r_0 and is unpredictable. To eliminate the use of the second key and still satisfy the requirements for z_0 suggested above, we can compute $z_0 = f(r_0 + 1)$ in stateless implementations, whereas in stateful implementations we compute $z_0 = IV + r_0$, where the per-message r_0 can be generated as a random value, or as an encryption of *ctr* in the XORC mode. This eliminates the additional block-cipher invocations necessary in the stateless and stateful-sender modes at the cost of maintaining an extra shared state variable (IV). This choice still satisfies the requirements for z_0.

Generalization. The above method for protecting message integrity against existential forgeries in chosen-plaintext attacks can be generalized as follows. Let the output ciphertext y_i be computed as $y_i = z_i \; op \; E_i$, where *op* is the ran-

[3] As a simple example illustrating why this is the case, let $z_0 = r_0 + 1$, choose x_1 such that $z_0 \oplus x_1 = r_0$ with non-negligible probability, and then compute $y_1 - y_0 = r_0$. With a known r_0, one can cause collisions in the values of z_i and break integrity.

domization operation, E_i are the elements of the randomization sequence, and z_i the hidden ciphertext generated by the encryption mode Π. The encryption mode Π (1) must be secure against adaptive chosen-plaintext attacks with respect to secrecy, and (2) must use the input plaintext blocks x_i to generate the input to f. The PCBC [13,16], and the "infinite garble extension" [5] modes are suitable, but counter-mode/XORC and XOR$ are not (since they fail condition (2)). Operation *op* must be invertible, so $\oplus$, modular 2^l addition and subtraction are appropriate. Elements E_i must be unpredictable such that collisions among z_i's (discussed above) could only occur with negligible probability. Other sequences can be used. For example $E_i = a^i \times r_0$ can be used, where E_i is a linear congruence sequence with multiplier a, where a can be chosen so that the sequence passes spectral tests to whatever degree of accuracy is deemed necessary. (Examples of good multipliers are readily available in the literature [12].)

XCBC-XOR Modes. To illustrate the properties of the XCBC modes in integrity attacks, we choose $g(x) = z_0 \oplus x_1 \oplus \cdots \oplus x_n$ for plaintext $x = x_1 \cdots x_n$, where z_0 is defined as the initialization vector of the mode. In this example, block $g(x)$ is appended to the *end* of a n-block message plaintext x, and hence block $x_{n+1} = z_0 \oplus x_1 \oplus \cdots \oplus x_n$. For this choice of $g(x)$, the integrity check performed at decryption becomes $z_0 \oplus x_1 \oplus \cdots \oplus x_n = f^{-1}(z_{n+1}) \oplus z_n$, where $z_{n+1} = y_{n+1} - (n+1) \times r_0$, and $z_n = y_n - n \times r_0$.

In the XCBC modes padding is done with a standard pattern that always starts with a "1" bit followed by the minimum number of "0" bits necessary to fill the last block of plaintext. If the last block of a message does not need padding, use block $g'(x) = \overline{z_0} \oplus x_1 \oplus \cdots \oplus x_n$ as the x_{n+1} plaintext block, where $\overline{z_0}$ is the bitwise complement of z_0; otherwise, use $g(x) = z_0 \oplus x_1 \oplus \cdots \oplus x_n$. This avoids the extra block encryption which would otherwise be necessary for plaintexts consisting of an integral number of blocks.

The stateless and stateful encryption modes Π-g obtained by the use of schemes Π = XCBC$, Π = XCBCC, or Π = XCBCS with function $g(x) = z_0 \oplus x_1 \oplus \cdots \oplus x_n$ are denoted by XCBC$-$XOR$, XCBCC-$XOR$, and XCBCS-XOR respectively.

Examples of Other Encryption Modes that Preserve Message Integrity.

Katz and Yung [11] proposed an interesting single-pass encryption mode, called the Related Plaintext Chaining (RPC), that is EF-CPA secure when using a non-cryptographic MDC function g consisting only of message start and end tokens. RPC has several important operational advantages, such as full parallelization, incremental updates, out-of-order processing, and low upper bound on the probability of adversary's success in producing a forgery. However, it wastes a substantial amount of throughput since it encrypts the block sequence number and message data in the same block. This may make the selection of modern hash functions as the MDC function g for common encryption modes, such as CBC, a superior performance alternative, at least for sequential implementations. Similarly the use of modern MACs, such as the UMAC, with a separate

key may also produce better overall throughput performance than RPC when used with common encryption modes.

Recently, C.S. Jutla [14] proposed an interesting scheme in which the output blocks z_i of CBC encryption are modified by (i.e., bitwise exclusive-or operations) with a sequence E_i of pairwise independent elements. In this model, $E_i = (i \times IV_1 + IV_2) mod\ p$, where IV_1, IV_2 are random values generated from an initial random value r, and p is prime, and the complexity is $n + 3$, where n is the length of the plaintext input in blocks. In contrast with C.S. Jutla's scheme, the elements of the XCBC sequence, $E_i = (i \times r_0) mod\ 2^l$, are not pairwise independent, and the complexity is $n + 2$ for the stateless and stateful-sender cases, and $n + 1$ for the stateful case. Also, the performance of the required modular 2^l additions is slightly better than that of *mod p* additions, where p is prime. However, the pairwise independence of C.S. Jutla's E_i sequence should yield a slightly tighter bound on the probability of successful forgery illustrating, yet again, a fundamental tradeoff between performance and security. (The bound is tighter by a fraction of a $\log_2$ factor depending on the value of p, which would mean that the attack complexity is within the same order of magnitude of the XCBC bound – viz., Section 5).

More recently, P. Rogaway [20] has proposed other schemes that use interesting variations of non-independent and pairwise-independent elements for the E_i sequence, similar to the sequences presented in this paper and C.S. Jutla's, to achieve $n+1$ complexity. Under the same assumptions regarding stateful and stateless implementations, C.S. Jutla's modes require an extra block enciphering over the XCBC and P. Rogaway's modes. We note that all modes for authenticated encryption include an extra block cipher operation for the enciphering of the exclusive-or MDC; e.g., stateful XCBC and P. Rogaway's OCB mode, which is also stateful, require $n + 2$ block-cipher invocations.

Architecture-Independent Parallel Encryption. C.S. Jutla's recent parallel mode [14] requires that both the input to and output of the block cipher are randomized using a sequence of *pairwise-independent* random blocks. Our fully parallel modes achieve the same effect *without* using a sequence of pairwise-independent random blocks. For these modes, it is sufficient to randomize the input and output blocks of f using the same type of sequence. In this case, the probability of input or output collisions, which would be necessary to break security and integrity respectively, would remain negligible. An example is the *stateful Extended Electronic Codebook-XOR* encryption (XECBS-*XOR*) mode, in which for index $i, 1 \leq i \leq n+1, n = |x|$, the ciphertext block y_i is obtained through the formulae:

$$y_i = f(x_i + ctr \times R + i \times R^*) + ctr \times R + i \times R^*, \quad \forall i, 1 \leq i \leq n, ctr \leq q_e$$
$$y_{n+1} = f(x_{n+1} + ctr \times R) + ctr \times R + (n+1) \times R^*,$$

where R, R^* are two random, uniformly distributed and independent blocks each of l bits in length that are part of the keying state shared by the sender and receiver, and *ctr* is the counter that serves as message identifier. The counter *ctr* is initialized to 1 and increased by 1 on every message encryption up to

q_e, which is the bound of the number of allowable message encryptions (viz., Theorem 4 below). Note that a per-message unpredictable or random nonce is unnecessary, and that the sequence of elements $E_i = ctr \times R + i \times R^*$ can be precomputed for multiple messages, can be computed incrementally [4], and in an out-of-order manner. To provide authentication, the last block is computed using the following formula for the function g:

$$x_{n+1} = g(x) = x_1 \oplus \cdots \oplus x_n.$$

This authenticated encryption mode achieves optimal performance; i.e., $n + 1$ parallel block cipher invocations, and has a throughput close to that of a single block-cipher invocation. The security of the XECBS-XOR mode with respect to confidentiality in an adaptive chosen-plaintext attack can be demonstrated in the same manner as that used for the CBC mode [1].

For the XECBS-XOR encryption scheme proposed above, padding follows the similar conventions as those for the XCBC-XOR modes to distinguish between padded and unpadded messages; i.e., the following formula is used for the enciphering of the last block.

$$y_{n+1} = f(x_{n+1} + ctr \times Z) + ctr \times R + (n + 1) \times R^*,$$

where $Z = \overline{R}$ is the bitwise complement of R and is used for unpadded messages, and $Z = R$ for padded messages.

Stateless architecture-independent parallel modes and stateful-sender architecture-independent parallel modes can be specified in the same manner as those for the XCBC modes. For example, for the stateless mode, an l-bit random number number r_0 is generated and one can replace $ctr \times R$ with r_0 and use $R^* = f(r_0 + 1)$ in the formulae for the stateless encryption mode; r_0 is also enciphered to generate $y_0 = f(r_0)$ which is part of the ciphertext string. In the stateful-sender mode, $r_0 = f(ctr)$, where ctr is an l-bit counter initialized to a constant such as -1; one can replace $ctr \times R$ with r_0 use $R^* = f(r_0)$ in the formulae for the stateful-sender encryption mode. In the modes thus obtained (and other related variants), there would not be any ciphertext chaining, and a priori knowledge of the number of processors would be unnecessary.

As noted earlier, the sequence $E_i = ctr \times R + i \times R^*$ does not completely hide the low order bits of x_i thereby enabling verification of key guesses by an adversary. Resistance to such attacks can be implemented in a similar manner as that of DESX [19], if deemed necessary. However, adoption of modern block ciphers with long keys should reduce the need for this.

4 Definition of the XECB Authentication Modes

In this section, we introduce new Message Authentication Modes (MACs) that counter adaptive chosen-message attacks [2]. We call these MACs the eXtended Electronic Codebook MACs, or XECB-MACs. The XECB-MAC modes have all the properties of the XOR MACs [2], but they do not waste half of the block

size for recording the block identifier thereby avoid doubling the number of block cipher invocations. Many variants of XECB-MACs are possible, and here we present stateless version, XECB$-MAC, a stateful-sender version XECBC-MAC, and a stateful version, the XECBS-MAC.

Message Signing. In both the stateless and stateful-sender implementation, we generate a per-message random value y_0 that is used to randomize each plaintext block of a message x, namely $x_i, 1 \leq i \leq n, n = |x|$, before it is fed to the block cipher function f, where $f = F_K$ is selected from a PRF family F by a key K, which is random and uniform. The result of the randomization is $x_i + i \times y_0$, and the result of block enciphering with f is $y_i = f(x_i + i \times y_0)$. The stateless mode initialization requires a random number generator to create the random block r_0; i.e., $r_0 \leftarrow \{0,1\}^l$. Then $y_0 = f(r_0)$. Stateful-sender implementations avoid the use of the random number generator, and instead, uses a counter ctr, to create y_0 directly, namely $y_0 = f(ctr)$. The counter ctr is initialized by the sender on a per-key basis to a constant, such as -1, and is maintained across consecutive signing requests for the same key K.

For the purposes of simplifying the proofs, we made the following choices for the generation and use of random vector z_0 in both implementations: (1) an additional per-message unpredictable block z_0 is generated and treated as an additional last block of the message plaintext before it is also randomized and enciphered by f, namely $x_{n+1} = z_0$ and $y_{n+1} = f(z_0 + (n+1) \times y_0)$; and (2) we set $z_0 = f'(r_0)$, where $f' = F_{K'}$ is a PRF selected with the second key K'. Clearly, the generation of z_0 can be performed with the same key, K, by block enciphering a simple function of r_0 (e.g., $f(r_0+1)$), and use of K' becomes unnecessary.

The block cipher outputs, $y_1, \cdots, y_n, y_{n+1}$, are exclusive-or-ed to generate the authentication tag $w = y_1 \oplus \cdots \oplus y_n \oplus y_{n+1}$. The algorithm outputs the pair (r_0, w) in the stateless mode, and (ctr, w) in the stateful-sender mode.

We include the stateful-sender version of the XECB MAC modes below. For the (very similar) stateless version, the reader is referred to *http://csrc.nist.gov/encryption/modes/proposedmodes*.

Stateful-Sender XECB-MAC Mode (XECBC-MAC)

function Sign-XECBC-MAC$^f(ctr, x)$
$y_0 = f(ctr), z_0 = f'(y_0)$
$x_{n+1} = z_0$
for $i = 1, \cdots, n+1$ **do** {
$y_i = f(x_i + i \times y_0)$ }
$w = y_1 \oplus \cdots \oplus y_n \oplus y_{n+1}$
$ctr' \leftarrow ctr + 1$
return (ctr, w)

function Verify-XECBC-MAC$^f(x, ctr, w)$
$y_0 = f(ctr), z_0 = f'(y_0)$
$x_{n+1} = z_0$
for $i = 1, \cdots, n+1$ **do** {
$y_i = f(x_i + i \times y_0)$ }
$w' = y_1 \oplus \cdots \oplus y_n \oplus y_{n+1}$
if $w = w'$ **then return** 1
else return 0.

Note that ctr' represents the updated ctr value.

Message Verification. For verification, an adversary submits a forgery $x = x_1 \cdots x_n$ and a forged pair (r_0, w) or (ctr, w) depending upon the mode.[4] Message x is then signed and an authentication tag $w' = y_1 \oplus \cdots \oplus y_n \oplus y_{n+1}$ is generated. The algorithm outputs a bit that is either 1, if the forged authentication tag is correct, namely $w = w'$, or 0, otherwise.

Block-Cipher Invocations and Mode Throughput. The number of block-cipher invocations in the stateless and stateful-sender XECB modes can be reduced from $n + 3$ to $n + 2$ simply by eliminating the enciphering of block x_{n+1}. For example, the enciphering of the last plaintext block can be changed to $y_n = f(x_n \oplus z_0 + n \times y_0)$ (without affecting the proofs significantly). Furthermore, in a stateful version, a random variable R is maintained on the per-key basis, and $z_0 = R + y_0$, as in the XCBCS encryption mode. This would eliminate the extra block enciphering of the function of r_0 for each message, without affecting the proofs. Hence, in a stateful XECB MAC mode, the number of block-cipher invocations becomes $n + 1$, which is one more than that of PMAC [21], which is also a stateful mode. As is the case with PMAC, the throughput of the XECB modes comes close to that corresponding to two sequential block-cipher invocations (as opposed to that of XOR-MAC, which corresponds to that of a single block-cipher invocation).

The following stateful variant of the XECB modes appears to come close to the optimal performance of any parallel MAC, namely n parallel block-cipher invocations and throughput equivalent of a single block-cipher invocation.

Stateful XECB-MAC Mode (XECBS-MAC)
Let R, R^* be two random, uniformly distributed and independent blocks that are part of the keying state shared by the sender and receiver.

function Sign-XECBS-MAC$^f(ctr, x)$
for $i = 1, \cdots, n$ **do** {
$y_i = f(x_i + ctr \times R + i \times R^*)$ }
$w = y_1 \oplus \cdots \oplus y_n$
$ctr' \leftarrow ctr + 1$
return (ctr, w)

function Verify-XECBS-MAC$^f(x, ctr, w)$
if $ctr > q_s$ **then return** 0
for $i = 1, \cdots, n$ **do** {
$y_i = f(x_i + ctr \times R + i \times R^*)$ }
$w' = y_1 \oplus \cdots \oplus y_n$
if $w = w'$ **then return** 1
else return 0.

Note that ctr is initialized to 1, and ctr' represents the updated ctr value.

For the XECBS-MAC mode proposed above, padding follows the similar conventions as those for the XECBS-XOR mode to distinguish between padded and unpadded messages; i.e., the following formula is used for the enciphering of the last block.

$$y_n = f(x_n + ctr \times Z + n \times R^*),$$

where $Z = \overline{R}$ is the bitwise complement of R and is used for unpadded messages, and $Z = R$ for padded messages. Note that, as in the case of the XECBS-XOR

[4] The forgeries (x, r_0, w) or (x, ctr, w) are not previously signed queries. Note also that the length n of the forged message need not be equal to the length of any signed message.

mode, a per-message unpredictable or random nonce is unnecessary, and that the sequence of elements $E_i = ctr \times R + i \times R^*$ can be precomputed for multiple messages, can be computed incrementally, and in an out-of-order manner.

Properties of the XECB Authentication Modes

1. Security. The XECB authentication modes are intended to be secure against adaptive chosen-message attacks [2]. Theorem 2 below shows the security bounds for the stateful-sender mode. The XECB modes, as well as all the other modes that use similar types of randomization sequences, have higher, but still negligible, upper bounds on the adversary's success in producing a forgery than those of the XOR-MAC modes.

2. Parallel and Pipelined Operation. Block-cipher (e.g., AES) computations on different blocks can be made in a *fully parallel* or *pipelined* manner; i.e., it can exploit any degree of parallelism or pipelining available at the sender or receiver without a priori knowledge of the number of processors available.

3. Incremental Updates. The XECB-MAC modes are incremental with respect to block replacement; e.g., a block x_i of a long message is replaced with a new value x'_i. For instance, let us consider the stateful-sender mode. Let the two messages have the same counter *ctr*; hence, the authentication tag of the new message, w', is obtained from the authentication tag of the previous message, w, by the following formula: $w' = w \oplus f(x_i + i \times y_0) \oplus f(x'_i + i \times y_0)$. The replacement property can be easily extended to insertion and deletion of blocks.

4. Out-of-order Verification. The verification of the authentication tag can proceed even if the blocks of the message arrive out of order as long as each block is accompanied by its index and the first block has been retrieved.

5 Security Considerations

The theorems and proofs that demonstrate that these modes are secure with respect to secrecy (e.g., in a left-or-right sense) are similar to those of the CBC mode [1] and, therefore, are omitted. For an XCBC mode, we can determine the $(t', q', \mu', \epsilon')$ secrecy parameters; i.e., an adversary making at most q' queries, totaling at most μ' bits, and taking time t' has an advantage in breaking the secrecy of that that mode (e.g., in a left-or-right sense) that is bounded by a negligible ϵ'.

In establishing the security of the XCBC$ mode against the message-integrity attack, let the parameters used in the attack be bound as follows: $q_e \leq q'$, since the XCBC$ mode is also chosen-plaintext secure, $t_e + t_v \leq t$, and $\mu'' = \mu_e + \mu_v \leq ql$. Let the forgery verification parameters q_v, μ_v, t_v be chosen within the constraints of these bounds and to obtain the desired $Pr_{f \overset{\mathcal{R}}{\leftarrow} F}[Succ]$.

Theorem 1 [Security of XCBC$-$XOR$ against a Message-Integrity Attack].

Suppose F is a (t, q, ϵ)-secure SPRP family with block length l. The mode XCBC$-$XOR$ is secure against a message-integrity attack consisting of $q_e + q_v$ queries, totaling $\mu_e + \mu_v \leq ql$ bits, and taking at most $t_e + t_v \leq t$ time; i.e., the probability of adversary's success is

$$Pr_{f \stackrel{\mathcal{R}}{\leftarrow} F}[\text{Succ}] \leq \epsilon + \frac{\mu_v(\mu_v - l)}{l^2 2^{l+1}} + \frac{q_e(q_e - 1)}{2^{l+1}} + \frac{(q_e + 1)\mu_v}{l2^l} + \frac{\mu_v}{l2^{l+1}}(\log_2 \frac{\mu_v}{l} + 3) + \frac{q_v \mu_e}{l2^l}(\log_2 \frac{\mu_e}{l} + 3).$$

(The proof of Theorem 1 can be found in the full version of the paper available at *http://csrc.nist.gov/encryption/modes/proposedmodes*.) Note that parameters q_e, μ_e, t_e can be easily stated in terms of secrecy parameters $(t', q', \mu', \epsilon')$ above by introducing a constant c' defining the speed of the XOR function.

Theorem 1 above allows us to estimate the complexity of a message-integrity attack.[5] In a successful attack, $Pr_{f \stackrel{\mathcal{R}}{\leftarrow} F}[\text{Succ}] \in (\text{negligible}, 1]$. To estimate complexity, we set the probability of success when $f \stackrel{\mathcal{R}}{\leftarrow} P^l$ to the customary 1/2, and assume that the attack parameters used in the above bound, namely $\frac{\mu_e}{l}, \frac{\mu_v}{l}$, are of the same order or magnitude, namely $2^{\alpha l}$, where $0 < \alpha < 1$. Also, since the shortest message has at least three blocks, $q_e, q_v \leq \lfloor \frac{2^{\alpha l}}{3} \rfloor$. In this case, by setting

$$\frac{q_e(q_e - 1)}{2^{l+1}} + \frac{\mu_v(\mu_v - l)}{l^2 2^{l+1}} + \frac{(q_e + 1)\mu_v}{l2^l} + \frac{\mu_v}{l2^{l+1}}(\log_2 \frac{\mu_v}{l} + 3) + \frac{q_v \mu_e}{l2^l}(\log_2 \frac{\mu_e}{l} + 3) = 1/2,$$

we obtain (by ignoring the $\lfloor . \rfloor$ function) the equation $2^{2\alpha l} \frac{6\alpha l + 34}{9} + 2^{\alpha l} \frac{3\alpha l + 11}{3} = 2^l$, which allows us to estimate α for different values of l. (In this estimate, we can ignore the term in $2^{\alpha l}$ since it is insignificant compared to the other term of the sum.) For example, for $l = 64, \alpha \approx \frac{29}{64}$, for $l = 128, \alpha \approx \frac{61}{128}$, and for $l = 256, \alpha \approx \frac{124}{256}$. Hence, this attack is very close to a square-root attack (i.e., $\alpha \to \frac{1}{2}$ as l increases), and remains this way even is the secrecy bound of Lemma 1 presented in the full version of the paper available at *http://csrc.nist.gov/encryption/modes/proposedmodes* (adjusted for PRPs) is taken into account for integrity (i.e., half of it is added to the integrity bound). Thus the payoff from improved bounds using families of SPRPs is limited.

A variant of Theorem 1 can be proved for the stateful modes. Furthermore, similar theorems hold for single-key stateless modes. The statement and proof for such theorems are similar to the statement and proof for the integrity theorem for the stateless mode, and hence, are omitted.

[5] Technically, the complexity of a successful integrity attack, and the bound of Theorem 1, should account for the success of a secrecy attack; i.e., half of the secrecy bound shown of Lemma 1 presented in the full version of the paper available at *http://csrc.nist.gov/encryption/modes/proposedmodes* (adjusted for the use of PRPs) should be added to the bound in Theorem 1. This is the case because, in general, in modes using the same key for both secrecy and integrity, a successful secrecy attack can break integrity and, vice-versa, a successful integrity attack can break secrecy. (This can be shown using the secrecy and integrity properties of the *IGE* mode; viz., *http://csrc.nist.gov/encryption/modes/proposedmodes*.) As suggested below, the addition of the secrecy bound would not affect the complexity of a successful integrity attack.

The XECB-MAC modes are intended to be secure against adaptive chosen-message attacks [2] consisting of up to q_s signature queries totaling at most μ_s bits and using time up to t_s, and q_v verification queries totaling at most μ_v bits and using time at most t_v. The security of the XECBC-MAC mode is established by the following theorem.

Theorem 2 [Security of XECBC-MAC in an Adaptive Chosen-Message Attack].

Suppose F is a (t, q, ϵ)-secure PRF family with block length l. The message authentication mode (Sign-XECBC-MAC^f, Verify-XECBC-MAC^f, KG) is secure against adaptive chosen-message (q_s, q_v) attacks consisting of $q_s + q_v$ queries totaling $\mu_s + \mu_v \leq ql$ bits and taking at most $t_s + t_v \leq t$ time; i.e., the probability of adversary's success is

$$Pr_{f \overset{\mathcal{R}}{\leftarrow} F}[\text{Succ}] \leq \epsilon + \frac{\mu_v}{l2^l}(\log_2 \frac{\mu_v}{l} + 3) + \frac{q_s \mu_v}{l2^l} + \left(q_s + 2q_v + \frac{\mu_s}{2l}\right) \frac{\mu_s}{l2^{l+1}}(\log_2 \frac{\mu_s}{l} + 3).$$

The proof of this theorem is similar to that of Theorem 1 and is presented in the full version of the paper available at
http://csrc.nist.gov/encryption/modes/proposedmodes.

A similar theorem can be provided for the stateless message authentication mode. The complexity of an attack against XECBC-MAC can be determined in a similar manner to that of an attack against the XCBC$-XOR mode.

We also present a theorem for the security of the XECBS-MAC mode. (The restatement of this theorem in terms of a family of PRPs, such as AES, and the corresponding proof modifications are standard.)

Theorem 3 [Security of XECBS-MAC in an Adaptive Chosen-Message Attack].

Suppose F is a (t, q, ϵ)-secure PRF family with block length l. The message authentication mode (Sign-XECBS-MAC^f, Verify-XECBS-MAC^f, KG) is secure against adaptive chosen-message (q_s, q_v) attacks consisting of $q_s + q_v$ queries $(q_v \leq q_s)$ totaling $\mu_s + \mu_v \leq ql$ bits and taking at most $t_s + t_v \leq t$ time; i.e., the probability of adversary's success is

$$\begin{aligned} Pr_{f \overset{\mathcal{R}}{\leftarrow} F}[\text{Succ}] \leq \epsilon + \frac{q_v}{2^l} + \frac{\mu_v}{l2^{l+1}}(\log_2 \frac{\mu_v}{l} + 3) + \left(q_v + \frac{\mu_s}{l}\right) \frac{q_s}{2^{l+1}}(\log_2 q_s + 3) \\ + \left(q_v + \frac{\mu_s}{l}\right) \frac{\mu_s}{l2^{l+1}}(\log_2 \frac{\mu_s}{l} + 3). \end{aligned}$$

The proof of this theorem is similar to that of Theorems 1 and 2 and is presented in the full version of the paper available at
http://csrc.nist.gov/encryption/modes/proposedmodes.

The security of the XECBS-XOR mode in a message-integrity attack is shown by the theorem bellow.

Theorem 4 [Security of XECBS-XOR in a Message-Integrity Attack].

Suppose F is a (t, q, ϵ)-secure SPRP family with block length l. The mode XECBS-XOR is secure against a message-integrity attack consisting of $q_e + q_v$

queries ($q_v \leq q_e$), totaling $\mu_e + \mu_v \leq ql$ bits, and taking at most $t_e + t_v \leq t$ time; i.e., the probability of adversary's success is

$$Pr_{f \xleftarrow{R} F}[\text{Succ}] \leq \epsilon + \frac{\mu_v(\mu_v - l)}{l^2 2^{l+1}} + \frac{q_v}{2^l} + \frac{\mu_v}{l2^{l+1}}(\log_2 \frac{\mu_v}{l} + 3) + \frac{\mu_e(\mu_e - l)}{l^2 2^{l+1}} + \left(q_v + \frac{\mu_e}{l}\right) \frac{q_e}{2^{l+1}}(\log_2 q_e + 3) + \left(q_v + \frac{\mu_e}{l}\right) \frac{\mu_e}{l2^{l+1}}(\log_2 \frac{\mu_e}{l} + 3).$$

(The proof of Theorem 4 can be found in the full version of the paper available at *http://csrc.nist.gov/encryption/modes/proposedmodes*.) Note that maximum allowable values for q_s and q_e in Theorems 3 and 4 can be determined by setting the probability of successful forgery to a desired value.

Acknowledgments. We thank David Wagner for pointing out an oversight in an earlier version of Theorem 1, Tal Malkin for her thoughtful comments and suggestions, Omer Horvitz and Radostina Koleva for their careful reading of this paper.

References

1. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A Concrete Security Treatment of Symmetric Encryption," Proceedings of the 38th Symposium on Foundations of Computer Science, IEEE, 1997, (394-403). A full version of this paper is available at *http://www-cse.ucsd.edu/users/mihir*.
2. M. Bellare, R. Guerin, and P.Rogaway, "XOR MACs: New methods for message authentication using finite pseudo-random functions", Advances in Cryptology-CRYPTO '95 (LNCS 963), 15-28, 1995. (Also U.S. Patent No. 5,757,913, May 1998, and U.S. Patent No. 5,673,318, Sept. 1997.)
3. M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm," manuscript, May 26, 2000. *http://eprint.iacr.org/2000.025.ps*.
4. E. Buonanno, J. Katz and M. Yung, "Incremental Unforgeable Encryption, " Proc. Fast Software Encryption 2001, M. Matsui (ed.) (to appear in Springer-Verlag, LNCS).
5. C.M. Campbell, "Design and Specification of Cryptographic Capabilities," in *Computer Security and the Data Encryption Standard*, (D.K. Brandstad (ed.)) National Bureau of Standards Special Publications 500-27, U.S. Department of Commerce, February 1978, pp. 54-66.
6. Open Software Foundation, "OSF - Distributed Computing Environment (DCE), Remote Procedure Call Mechanisms," Code Snapshot 3, Release, 1.0, March 17, 1991.
7. V.D. Gligor and B. G. Lindsay,"Object Migration and Authentication," *IEEE-Transactions on Software Engineering*, SE-5 Vol. 6, November 1979. (Also IBM-Research Report RJ 2298 (3l04), August 1978.)
8. V.D. Gligor, and P. Donescu, "Integrity-Aware PCBC Schemes," in Proc. of the 7th Int'l Workshop on *Security Protocols*, (B. Christianson, B.Crispo, and M. Roe (eds.)), Cambridge, U.K., LNCS 1796, April 2000.

9. R.R. Juneman, S.M. Mathias, and C.H. Meyer, "Message Authentication with Manipulation Detection Codes," Proc. of the IEEE Symp. on Security and Privacy, Oakland, CA., April 1983, pp. 33-54.
10. J. Katz and M. Yung, "Complete characterization of security notions for probabilistic private-key encryption," Proc. of the 32nd Annual Symp. on the Theory of Computing, ACM 2000.
11. J. Katz and M. Yung, "Unforgeable Encryption and Adaptively Secure Modes of Operation," Proc. Fast Software Encryption 2000, B. Schneir (ed.) (to appear in Springer-Verlag, LNCS).
12. D.E. Knuth, "The Art of Computer Programming - Volume 2: Seminumerical Algorithms," Addison-Wesley, 1981 (second edition), Chapter 3.
13. J. T. Kohl, "The use of encryption in Kerberos for network authentication", *Advances in Cryptology-CRYPTO '89* (LNCS 435), 35-43, 1990.
14. C.S. Jutla, "Encryption Modes with Almost Free Message Integrity," IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, manuscript, August 1, 2000. *http://eprint.iacr.org/2000/039.*
15. M Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions", *SIAM J. Computing*, Vol. 17, No. 2, April 1988.
16. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, 1997.
17. M. Naor and O. Reingold, "From Unpredictability to Indistinguishability: A Simple Construction of Pseudo-Random Functions from MACs," *Advances in Cryptology - CRYPTO '98* (LNCS 1462), 267-282, 1998.
18. RFC 1510, "The Kerberos network authentication service (V5)", Internet Request for Comments 1510, J. Kohl and B.C. Neuman, September 1993.
19. P. Rogaway, "The Security of DESX," RSA Laboratories *Cryptobytes*, Vol. 2, No. 2, Summer 1996.
20. P. Rogaway, "OCB Mode: Parallelizable Authenticated Encryption", Preliminary Draft, October 16, 2000, available at *http://csrc.nist.gov/encryption/aes/modes/rogaway-ocb1.pdf.*
21. P. Rogaway, "PMAC: A Parallelizable Message Authentication Mode," Preliminary Draft, October 16, 2000, available at *http://csrc.nist.gov/encryption/aes/modes/rogaway-pmac1.pdf.*
22. S. G. Stubblebine and V. D. Gligor, "On message integrity in cryptographic protocols", Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy, 85-104, 1992.

Incremental Unforgeable Encryption

Enrico Buonanno[1], Jonathan Katz[2], and Moti Yung[3]

[1] Department of Computer Science, Columbia University, NY, USA.
eb659@columbia.edu
[2] Telcordia Technologies, Inc., NJ, USA and
Department of Computer Science, Columbia University.
jkatz@cs.columbia.edu
[3] CertCo, Inc., USA.
moti@cs.columbia.edu

Abstract. The recent selection of the AES block cipher to replace DES has generated interest in developing new modes of operation to supplement the modes defined as part of the DES standard [1,16,23]. We initiate the study of modes of encryption which are both *incremental* and *unforgeable*, and point out a number of applications for modes meeting these requirements. We also propose three specific modes achieving these goals, and discuss the strengths and weaknesses of each.

1 Introduction

1.1 Motivation

With the recent selection of the proposed AES, there has been an intensification of study on all aspects of private-key cryptography. In particular, there has been much interest in the design and analysis of new modes of operation for private-key encryption (indeed, NIST recently held a workshop focusing on this topic [22]). It is important to note that just as no block cipher is "best" for all applications, the same holds true for modes of encryption (hence the number of different modes proposed). In fact, each application determines a different set of requirements for the encryption scheme to be used.

It is often required to maintain an encrypted copy of data which undergoes frequent, yet small, changes [4]. One example is a user revising a file who wants to maintain an encrypted copy at all times. Other examples include the maintenance of an encrypted database or table throughout the course of many update operations during which isolated entries change while the bulk of the data remains the same. These examples may be taking place in an environment such as an encrypted file system [9] in which the underlying data is constantly changing yet encrypted versions must always be stored.

In such cases, using an incremental encryption scheme can lead to huge efficiency gains. The goal of incremental cryptography, introduced by Bellare, Goldreich, and Goldwasser [3], is to design cryptographic algorithms whose output can be updated very efficiently when the underlying input changes. For example,

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 109–124, 2002.

say we have a document D, and have applied cryptographic transformation T to this document to generate $\mathsf{T}(D)$. The document is then modified via update operation M (where, for example, $M =$ "delete block i") to give new document D'. An incremental update algorithm IncT is one such that $\mathsf{IncT}(D, M, \mathsf{T}(D))$ outputs a valid cryptographic transformation of D'. Note that computation of IncT can be potentially much faster than recomputing $\mathsf{T}(D')$ from scratch. Ideally, the running time of IncT should depend on the type of modification only (and possibly the security parameter), but, in particular, should be independent of the size of the document $|D|$. Even when this cannot be achieved, one might hope for an incremental update operation which runs in time $O(\log |D|)$ instead of time $O(|D|)$ which is required for re-computation from scratch. When documents change frequently, dramatic efficiency improvements are possible.

In many of the aforementioned scenarios, incrementality is not enough. Additionally, one typically requires a guarantee of data integrity. Consider the case of an encrypted file system or remote database. The resulting ciphertext may be stored on an insecure medium (this is the reason for encryption in the first place), and an adversary may be able to modify the ciphertext as he chooses. A malicious attacker should be prevented from modifying the ciphertext so that it will appear correct when it is later decrypted by the user. Data integrity is useful as a means of virus protection [4] if applications and data are always checked for validity before being used. We model this requirement via *unforgeability* [18, 19] (also known as ciphertext integrity [7]). Briefly, a malicious adversary (who views a sequence of encryptions and incremental update operations) should be unable to generate *any* new ciphertext which decrypts to a valid plaintext. This is the strongest notion of integrity for the case of encryption.

1.2 Previous Work

The joint importance of incrementality and integrity in the context of cryptographic file systems has been recognized elsewhere [4]. However, [4] focuses primarily on MAC and hashing algorithms. Achieving these goals simultaneously for the case of *encryption* has not previously been considered.

INCREMENTAL ENCRYPTION. Prior work dealing with incremental cryptography has focused mainly on hashing, signing, and message authentication [3,4,6,10,11, 21]. We are aware of only one previous work dealing with (among other things) incremental encryption [4]; the scheme there is based on encrypting a description of the modification and appending it to the end of the current ciphertext. A comparison of our work to [4] is worthwhile:

- A formal definition of security for incremental encryption does not appear in [4]. We provide appropriate definitions here.
- To achieve incrementality in the non-amortized sense, the scheme of [4] is complex and relatively inefficient. It requires $O(\log |D|)$ block cipher evaluations even for simple updates, and results in ciphertext which is as much as

four times longer than the plaintext. The schemes presented here are simple to implement, can be updated using a constant number of block cipher evaluations, and (in some cases) reduce the ciphertext expansion.
– The scheme of [4] does not achieve any measure of integrity (this was not the focus of their work). They did not consider active attacks, only semantic security. The schemes presented here are unforgeable under the strongest definition of the term.
– The basic scheme of [4] is not oblivious in the sense of hiding the revision history of the document (see Section **??**). The schemes presented here are all oblivious, without requiring additional complexity.

In fairness, the notion of security considered in [4] is stricter than that which we consider here. Specifically, we allow the adversary to determine the *location* of modifications made (although an adversary cannot determine the nature of the change). In practice, we believe this is not a serious concern. Discussion of this point appears in Section 2.3.

Unforgeability. The importance of integrity in the context of private-key encryption has been recognized for some time. Besides being important in its own right, integrity implies security against chosen ciphertext attacks [19] (see also [7]). Recently, there have appeared a flurry of definitions, detailed analyses, and modes of encryption all intending to more carefully address this issue [19, 17,7,8,12,24,22,15]. We use the notion of *unforgeability* [18,19] (also known as ciphertext integrity [7]), which is the strongest notion of integrity. Under this definition, an adversary may observe a sequence of encryptions (and incremental updates in our case) yet should be unable to generate *any* new ciphertext which decrypts to valid plaintext. The formal definition appears in Section 2.4.

1.3 Summary of Results

We begin by presenting our definitions: a definition for incremental encryption, formal definitions of security (privacy and integrity) for the setting of incremental encryption, and a definition of obliviousness. We then introduce three modes of encryption achieving both incrementality and unforgeability, and give theorems indicating the exact security of each construction. We conclude with a discussion comparing the strengths and weaknesses of these modes.

2 Definitions

2.1 Notation

For probabilistic algorithm A, denote by $y \leftarrow A(x_1, x_2, \ldots)$ the experiment in which we generate random coins r for A and let y equal the output of $A(x_1, x_2, \ldots; r)$. Furthermore, let $\{A(x_1, x_2, \ldots)\}$ represent the probability distribution defined by execution of A on the specified input, with coins for A generated randomly. If S is a set, then $b \leftarrow S$ denotes assigning to b an element uniformly chosen from S. If $p(x_1, x_2, \ldots)$ is a predicate, the notation

$\Pr[x_1 \leftarrow S; x_2 \leftarrow A(x_1, y_2, \ldots); \cdots : p(x_1, x_2, \ldots)]$ denotes the probability that $p(x_1, x_2, \ldots)$ is true after ordered execution of the listed experiments.

Strong Pseudorandom Permutations. We follow [13,2,20] in defining a strong pseudorandom permutation family F as one for which the input/output behavior of F_{sk}, F_{sk}^{-1} "looks random" to someone who does not know the randomly selected key sk. We refer to the listed references for details.

2.2 Incremental Encryption

Document Modifications. We view the document D as a sequence of blocks $\sigma_1, \ldots, \sigma_n$, where the blocksize may depend on the security parameter k. For simplicity, we assume that all documents consist of an integral number of blocks (documents can be padded using standard methods if this is not the case). In the context of incremental cryptography, various modification operations have been considered. We represent a generic modification operation by M and denote the effect of M on document D by $D\langle M\rangle$. We define sequential modifications operations by: $D\langle M_1, M_2, \ldots, M_k\rangle \stackrel{\text{def}}{=} (\cdots((D\langle M_1\rangle)\langle M_2\rangle)\cdots)\langle M_k\rangle$. We consider the following *types* of modification operations:

- $M = (\mathsf{delete}, i)$ deletes block i of the document.
- $M = (\mathsf{insert}, i, \sigma)$ inserts σ between the i^{th} and $(i+1)^{\text{th}}$ blocks of the document.
- $M = (\mathsf{replace}, i, \sigma)$ changes the i^{th} block of the document to σ.

Other modifications of interest include the cut and paste operations, which divide a document into two or combine two documents together. Although some of the schemes presented here support these modifications, we leave the details for a future version of this paper.

The *location* of modification operation M is the block number i which is modified. We always implicitly assume that a modification operation is valid; that is, it represents some feasible[1] modification of D. For the operations considered above, we define $|M|$ to be the underlying blocksize.

Incremental Encryption Schemes. We recall the generic definition of an incremental algorithm as presented by Bellare, Goldreich, and Goldwasser [4], modified here for the case of encryption. (Although the definition below explicitly mentions the security parameter k, we omit this parameter when discussing concrete security definitions and theorems since, in practice, we are given a fixed-size block cipher only.)

Definition 1. *An incremental, private-key encryption scheme Π defined over modification space $\mathcal{M}$ is a 4-tuple of algorithms $(\mathcal{K}, \mathcal{E}, \mathsf{IncE}, \mathcal{D})$ in which:*

- *$\mathcal{K}$, the* key generation algorithm*, is a probabilistic, poly(k)-time algorithm that takes as input security parameter k (in unary) and returns secret key sk. The security parameter also fixes a block size b.*

[1] E.g., we do not call (delete, i) on D when D contains fewer than i blocks.

- $\mathcal{E}$, *the* encryption algorithm, *is a probabilistic, poly*$(k, |D|)$*-time algorithm that takes as input sk and document* $D \in (\{0,1\}^b)^+$ *and returns ciphertext* C.
- IncE, *the* incremental update algorithm, *is a probabilistic, poly*$(k, |C|, |M|)$*-time algorithm that takes as input secret key sk, document* D, *modification operation* $M \in \mathcal{M}$, *and ciphertext* C *and returns modified ciphertext* C'.
- $\mathcal{D}$, *the* decryption algorithm, *is a deterministic, poly*$(k, |C|)$*-time algorithm that takes as input secret key sk and ciphertext* C *and returns either document* D *or a special symbol* $\perp$ *to indicate that ciphertext* C *is invalid.*

We require that for all sk which can be output by $\mathcal{K}$, *for all valid* D, *we have* $\mathcal{D}_{sk}(\mathcal{E}_{sk}(D)) = D$. *Additionally, for all sk which can be output by* $\mathcal{K}$, *for all valid* D, *for all modifications* $M \in \mathcal{M}$, *we have* $\mathcal{D}_{sk}(\mathsf{IncE}_{sk}(D, M, \mathcal{E}_{sk}(D))) = D\langle M \rangle$.

Remark 1. We do not define the behavior of $\mathsf{IncE}_{sk}(D, M, C)$ in the case when it is given "garbage" input; i.e., when $\mathcal{D}_{sk}(C) \neq D$ or $\mathcal{D}_{sk}(C) = \perp$. We refer to this sort of input as *invalid* for the incremental update algorithm.

Remark 2. It may seem strange that we allow the running time of IncE to be polynomial in $|D|$. However, such schemes may be of interest; for example, when an update can be done in half the time it would take to re-encrypt from scratch. Optimally, the running time of IncE should be independent of the length of document D. Following [4], such schemes are called *ideal*. In the present schemes, IncE does not require access to the original document D; thus, execution of the incremental update algorithm is abbreviated by $\mathsf{IncE}_{sk}(M, C)$.

Remark 3. The underlying modification space $\mathcal{M}$ is included in the specification. As discussed in Section 2.2, various modification operations can be considered; not every incremental encryption scheme supports every such operation.

2.3 Indistinguishability of Incremental Encryption

Although security for incremental encryption has been discussed informally in [4], this is the first formal definition of which we are aware. The secrecy requirements (informally) are as follows: first, the basic encryption algorithm should be semantically secure. Second, the incremental update algorithm should not somehow "ruin" the semantic security of the encryption. Finally, the incremental update algorithm itself should not leak information (discussed in more detail below) about the underlying modification.

According to the previous definition [4], an adversary, upon observing an incremental update, should not be able to determine the location of the modification taking place or the symbol being modified. We relax this requirement and allow the possibility that an adversary can determine where a modification takes place (but still cannot determine the symbol being modified). For example, an adversary should be unable to distinguish between a $(\mathsf{replace}, i, \sigma)$ and

a $(\mathsf{replace}, i, \sigma')$ modification. The case addressed by the previous definition is often not of practical concern (the location of a change may be known, anyway). Furthermore, the other benefits of our schemes outweigh this weaker security guarantee in many situations. In particular, our schemes are oblivious (hide details about the document modification history) and unforgeable, which are often more desirable goals. Finally, it is unclear how to extend the previous definition to handle modifications such as cut and paste, in which the "location" of the modification is trivially determined.

Formally, we model security using the notion of find-then-guess indistinguishability [2], which implies the standard notion [14] of semantic security. The adversary may interact with an *encryption oracle* $\mathcal{E}_{sk}(\cdot)$ and an *incremental update oracle* $\mathsf{IncE}_{sk}(\cdot,\cdot)$. Then, the adversary outputs either two documents (D_1, D_2) (which must have equal number of blocks ℓ) or a ciphertext C along with two modification operations (M_1, M_2) (which must be of the same type and modifying the same location). A bit b is chosen at random and kept hidden from the adversary. In the first case document D_b is encrypted, while in the second case the IncE algorithm is applied to M_b and C; in either case, the result is given to the adversary. The adversary may then continue to interact with the $\mathcal{E}_{sk}$ and IncE_{sk} oracles. We say the adversary *succeeds* if it correctly guesses b. We define the advantage of the adversary as twice the probability of success, minus $1/2$.

Our concrete security definition follows the approach employed by [5]. We say that encryption scheme Π is $(t, q_e, \mu, q_{inc}, \ell; \epsilon)$-secure in the sense of indistinguishability if, for any adversary A which runs in time t, making q_e queries to the encryption oracle and q_{inc} queries to the IncE oracle (with the total number of blocks in all encryption and incremental update queries equal to μ), the advantage of A is less than ϵ.

2.4 Unforgeability of Incremental Encryption

We now consider unforgeability for incremental modes of encryption, extending the definition of [19]. The adversary is again allowed to interact with an encryption oracle and an incremental update oracle. Now, however, we allow the adversary to submit only *valid* queries (in the sense of Remark 1) to the IncE algorithm. At the end of its execution, the adversary outputs a ciphertext C which must be different from any ciphertext it received from either of its oracles. The adversary succeeds if C is valid. Formally:

Definition 2. *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathsf{IncE}, \mathcal{D})$ be an incremental encryption scheme over modification space $\mathcal{M}$, and let A be an adversary. Let* $\mathsf{Adv}^{\mathrm{unf}}_{A,\Pi} \stackrel{\mathrm{def}}{=}$

$$\Pr\left[\mathrm{sk} \leftarrow \mathcal{K}; C \leftarrow A^{\mathcal{E}_{sk}(\cdot),\mathsf{IncE}_{sk}(\cdot,\cdot)} : \mathcal{D}_{sk}(C) \neq \perp\right].$$

We insist that A's queries to the IncE oracle are all valid, and that C was never received in response from either oracle. We say that Π is $(t, q_e, \mu, q_{inc}; \epsilon)$-secure in the sense of unforgeability if, for any adversary A which runs in time t, making

q_e queries to the encryption oracle and q_{inc} valid queries to the IncE *oracle (with the total number of blocks in all encryption and incremental update queries equal to μ),* $\mathsf{Adv}^{\mathrm{unf}}_{A,\Pi}$ *is less than ϵ.*

The above definition corresponds to the "basic security" of [3,4]. One may also consider the more complicated setting in which the adversary is given unrestricted access to the IncE oracle, and is allowed to submit invalid queries as well. Security in this setting is generally much more difficult to achieve [3,4,6]. In this paper, we concentrate on the case of basic security only[2] (which in typical applications is sufficient).

2.5 Obliviousness of Incremental Encryption

An incremental encryption scheme raises new security concerns. One such concern is that a ciphertext may reveal information about the revision history of the underlying plaintext. We say a scheme is *oblivious* if this revision history is hidden *even to someone who knows the secret key.* Motivation for this concern arises when the ciphertext is transmitted between two parties; the first party sending the ciphertext will not, in general, want the second party to be able to learn about the modifications made in the course of creating the document. This notion was first formally defined by Micciancio [21]; we modify his definition for the present context:

Definition 3. *Let Π be an incremental encryption scheme over modification space $\mathcal{M}$. We say that Π is* oblivious *if, for any two documents D, D', for any sequence of modifications $M_1, \ldots, M_i \in \mathcal{M}$ such that $D' = D\langle M_1, \ldots, M_i\rangle$, and for all keys sk, we have:*

$$\{\mathcal{E}_{sk}(D')\} \equiv \{\mathsf{IncE}_{sk}(M_i, \cdots, \mathsf{IncE}_{sk}(M_1, \mathcal{E}_{sk}(D)) \cdots)\}\,.$$

3 Incremental and Unforgeable Modes of Encryption

3.1 Encrypt-then-Incremental-MAC

One method of achieving both incrementality and unforgeability is to use an incremental mode of encryption together with an incremental MAC [3,4,6] of the ciphertext (the *encrypt-then-*MAC approach [7]). Let inc-$\mathcal{E}$ be an incremental encryption scheme, and let inc-MAC be a secure, incremental MAC algorithm. Then incremental, unforgeable encryption can be performed as follows:

$$\mathcal{E}_{k_1,k_2}(D) = C \circ \text{inc-MAC}_{k_2}(C), \tag{1}$$

where $C = \text{inc-}\mathcal{E}_{k_1}(D)$. Incremental updates are done in the straightforward way: first, perform the incremental update operation for C (based on the new

[2] The indistinguishability of our schemes, however, holds even when an adversary is given unrestricted access to the IncE oracle, as reflected in the definition.

document) to obtain C'; then, perform the incremental update operation for the MAC (based on the new ciphertext C').

Care must be taken with the details of inc-$\mathcal{E}$. First, note that not all incremental encryption algorithms will result in practical schemes. For example, if the incremental update operation for inc-$\mathcal{E}$ changes a large fraction[3] of the ciphertext C, there may not exist an efficient incremental update operation for the MAC. Furthermore, one must ensure that the incremental encryption scheme is indeed secure under the definition of Section 2.3. One possible solution is the following "randomized" ECB mode of encryption (rECB): given message $\sigma_1, \ldots, \sigma_n$, choose random $r_0, r_1, \ldots, r_n \leftarrow \{0,1\}^b$ (where b is the blocksize) and compute:

$$F_{sk}(r_0), F_{sk}(r_1 \oplus r_0), F_{sk}(\sigma_1 \oplus r_1), \ldots, F_{sk}(r_n \oplus r_0), F_{sk}(\sigma_n \oplus r_n).$$

This mode is a secure, incremental encryption scheme, with updates (replace, insert, and delete) done in the obvious way. Note that incremental updates result in only small changes to the ciphertext; thus, we can efficiently combine this mode with an incremental MAC algorithm (which handles replace, insert, and delete), as in (1). If the incremental MAC is secure, the result is a mode of encryption which is both incremental and unforgeable.

3.2 inc-IAPM Mode

The previous proposal is attractive for its simplicity. Unfortunately, in practice, incremental MAC algorithms are not very efficient (see Section 5). Thus, we propose other modes which improve the computational efficiency and ciphertext expansion rate.

The inc-IAPM mode described here is directly based on the IAPM mode of Jutla [17]. IAPM mode represents an advance over previous unforgeable modes of encryption, and is more efficient than the standard "encrypt-then-MAC" approach by a factor of two. Furthermore, it is parallelizable, which suggested to us the possibility that it could be adapted to give an incremental mode. However, it is non-trivial to modify this mode to achieve an *efficient*, incremental mode while completely satisfying our definitions of security.

We now present the details. Similarly to the IAPM mode, encrypting an n-block plaintext requires a sequence of pairwise independent blocks (labeled $S_0, S_1^\ell, S_1^r, \ldots, S_{n+1}^\ell, S_{n+1}^r, S^*$; see below) which will be used for "output whitening". These blocks are generated from a random seed K which is included with the ciphertext. Generation of these output whitening blocks is described in [17], and a more detailed analysis of the output whitening appears in [15]. It is important to note that generation of these blocks can be done very efficiently using $\mathcal{O}(\log n)$ block cipher calls [17] or even without using a block cipher at all [15]. In the analysis of our construction, we assume that blocks generated from the same seed are pairwise independent and blocks generated from different seeds are completely independent (this can be refined along the lines of [15]).

[3] Recall that an incremental encryption scheme might require time $O(|D|)$ for update operations, as long as this time is less than that required to re-encrypt from scratch.

Let F be a block cipher family with blocksize b. The key generation algorithm outputs a random key sk for F (in addition to the key necessary for generation of the whitening sequence). The plaintext is parsed as a sequence of b-bit blocks $\sigma_1, \ldots, \sigma_n$. Encryption is described in Fig. 1 (generation of the whitening sequence from seed K is as described above). Decryption is done by decrypt-

Algorithm $\mathcal{E}_{sk,sk'}(D; K)$
Generate $S_0, \ldots$ using K and sk'
for $i = 1$ to n :
$L_i \leftarrow \{0,1\}^b; R_i = L_i \oplus \sigma_i$
$L_0, R_0 \leftarrow \{0,1\}^b$
$L_{n+1} = \bigoplus_{i=0}^{n} L_i$
$R_{n+1} = \bigoplus_{i=0}^{n} R_i$
$\sigma^* = L_0 \oplus R_0$
for $i = 0$ to $n+1$:
$C_i^\ell = F_{sk}(L_i \oplus S_0^\ell) \oplus S_i^\ell$
$C_i^r = F_{sk}(R_i) \oplus S_i^r$
$C^* = F_{sk}(\sigma^* \oplus S^*) \oplus S_0$
return $K, C_0^\ell, C_0^r, \ldots, C_{n+1}^\ell, C_{n+1}^r, C^*$

Fig. 1. inc-IAPM mode of encryption.

ing each ciphertext block, re-generating the pairwise independent sequence, and checking the integrity conditions on L_{n+1}, R_{n+1}, and σ^*. If the integrity checks succeed, D_i is computed as $L_i \oplus R_i$; if they fail, the output is $\perp$.

For this particular mode, we are only able to achieve security with respect to the replace operation[4]. The incremental (replace, i, σ) operation proceeds as follows. First, blocks L_i and R_i are updated to reflect the new value of block i. In addition, new random values L_0 and R_0 are selected. Finally, in order to satisfy the integrity check, the algorithm updates the values for L_{n+1}, R_{n+1}, and σ^*. Details follow; for simplicity, we describe the algorithm informally (the ciphertext is parsed as a sequence of b-bit blocks $K, C_0^\ell, C_0^r, \ldots, C^*$):

Algorithm $\mathsf{IncE}_{sk,sk'}((\mathsf{replace}, i, \sigma), C)$
Compute $S_0, S_0^\ell, S_0^r, S_i^\ell, S_i^r, S_{n+1}^\ell, S_{n+1}^r, S^*$ from K
Decrypt to obtain $L_0, R_0, L_i, R_i, L_{n+1}, R_{n+1}$, and σ^*
Choose random L_0', L_i', and R_0'
Set $R_i' = L_i \oplus \sigma$
Update L_{n+1}', R_{n+1}', and σ'^* to satisfy the integrity checks
Re-encrypt $L_0', R_0', L_i', R_i', L_{n+1}', R_{n+1}'$, and σ'^*

[4] Incremental delete and insert modifications are known to be significantly more difficult to achieve in general [3,6].

3.3 RPC Mode

The inc-IAPM mode is more efficient than the encrypt-then-incremental-MAC approach, yet it only supports incremental replace operations. Here, we introduce RPC mode which is slightly less efficient than inc-IAPM, but supports replace, insert, and delete. The mode is specified by parameters b and r, where b is the block size and r is the amount of random padding. The document D is parsed as a sequence of $b-2r$-bit blocks $\sigma_1, \ldots, \sigma_n$. Encryption is performed as follows (start is not part of the valid message space):

Algorithm $\mathcal{E}_{sk}^{b,r}(D)$
for $i = 0$ to n :
$r_i \leftarrow \{0,1\}^r$
$C_0 = F_{sk}(r_0, \mathsf{start}, r_1)$
for $i = 1$ to $n-1$:
$C_i = F_{sk}(r_i, \sigma_i, r_{i+1})$
$C_n = F_{sk}(r_n, \sigma_n, r_0)$
$r^* = \bigoplus_{i=1}^{n} r_i$
$C^* = F_{sk}(r^* \oplus r_0, 0^{b-2r}, r^*)$
return $C_0 \ldots C_n C^*$

Decryption is done in the obvious way, by computing F_{sk}^{-1} for each block of the ciphertext and then checking that the first block contains an encryption of start, that the values $\{r_i\}$ are chained correctly, and that decryption of C^* gives the correct r_0 and checksum. If these integrity checks succeed, the computed document is output; if they fail, the output is $\perp$.

For lack of space, we describe (informally) the incremental delete, insert, and replace algorithms (a detailed description of these algorithms will appear in the full paper): The block i to be modified and adjacent blocks are decrypted to determine r_{i-1}, r_i, and r_{i+1}. When a new symbol σ is to be placed in position j (where $j = i, i+1$), σ is encrypted by choosing new, random r'_j and computing $C'_j = F_{sk}(r'_j, \sigma, r_{i+1})$. This causes the $\{r_i\}$ to remain chained correctly. Furthermore (and this occurs even during a delete modification), new r_0 and r_1 are chosen at random, and the checksum r^* is recomputed (and the resulting blocks are re-encrypted to give modified ciphertext blocks C'_0, C'_1, and C'^*).

4 Proofs of Security

The obliviousness of each mode is clear, by inspection. Thus, we focus on indistinguishability and unforgeability. We assume throughout this section that the block cipher family F is a family of strong pseudorandom permutations. The constructions can thus be analyzed by viewing F as a truly random permutation on b bits; "real-world" security bounds can be derived from the theorems below and exact security bounds on F. The theorems below give rough bounds on the security of the constructions. More detailed proofs, and tighter security bounds, will appear in the full version of this paper.

4.1 Encrypt-then-Incremental-MAC

A general composition theorem (along the lines of [7]) shows that a secure, incremental encryption scheme appended by a secure[5], incremental MAC of the result gives a secure, incremental encryption scheme which is also unforgeable. For the concrete security analysis, however, we choose the randomized ECB (rECB) mode (see Section 3.1) as the encryption algorithm, and the XOR-MAC of [4] (using r bits of random padding per block) as the MAC algorithm.

Theorem 1. rECB-XOR *over modification space* $\mathcal{M} = \{\mathsf{replace}, \mathsf{delete}, \mathsf{insert}\}$ *is* $(t, q_e, \mu, q_{inc}, \ell; \epsilon)$*-secure in the sense of indistinguishability, where:*

$$\epsilon = \mathcal{O}\left(\frac{\ell^2 + \mu\ell}{2^b} + \frac{\ell^2\mu^2}{2^{2b}}\right).$$

Proof (Sketch) Indistinguishability of the scheme is determined by rECB alone (since the MAC is computed on the ciphertext, it can be simulated by the adversary). Consider all blocks used as input to F during the course of the encryption oracle calls and incremental update oracle calls. Call such blocks *used*. First consider the case where the adversary outputs two documents, each containing ℓ blocks. Let r_0 and $\mathcal{B} = \{r_1 \oplus r_0, \sigma_1 \oplus r_1, \ldots, r_\ell \oplus r_0, \sigma_\ell \oplus r_\ell\}$ (cf. Section 3.1) be the set of blocks used as input to F during the encryption of the chosen document. The adversary has non-zero advantage in only the following cases: (1) For some $B_1, B_2 \in \mathcal{B}$, it is the case that $B_1 = B_2$; (2) for some i, $r_i \oplus r_0$, and $\sigma_i \oplus r_i$ are both *used*. The probability of (1) is bound by $\mathcal{O}(\ell^2/2^b)$, while the probability of (2) is bound by $\mathcal{O}(\ell^2\mu^2/2^{2b})$.

When the adversary outputs two modifications, the adversary has a non-zero advantage only if the modifications were of type replace or insert. In either case, a new, random r_i is chosen. The adversary can have non-zero advantage only if either (1) $r_i \oplus r_0$ is equal to $\sigma \oplus r_i$; or (2) $r_i \oplus r_0$ and $\sigma \oplus r_i$ are both *used*. The probability of either of these events is bound by $\mathcal{O}(\mu/2^b)$. ■

Theorem 2. rECB-XOR *over modification space* $\mathcal{M} = \{\mathsf{replace}, \mathsf{delete}, \mathsf{insert}\}$ *is* $(t, q_e, \mu, q_{inc}; \epsilon)$*-secure in the sense of unforgeability, where:*

$$\epsilon = \mathcal{O}\left(\frac{(q_e + q_{inc})^2}{2^b} + \frac{\mu^2}{2^r}\right).$$

Proof Unforgeability of the scheme is determined by the unforgeability of the MAC; taking the bounds from Theorem 3.1 of [4] gives the desired result. ■

[5] The security of the MAC must be that it is infeasible to forge a new, valid *pair* (M, tag); it is not sufficient that it be infeasible to forge a valid tag on a new message.

4.2 inc-IAPM

Theorem 3. *The inc-IAPM mode of encryption over modification space* $\mathcal{M} = \{\mathsf{replace}\}$ *is* $(t, q_e, \mu, q_{inc}, \ell; \epsilon)$*-secure in the sense of indistinguishability, where:*

$$\epsilon = \mathcal{O}\left(\frac{q_e + q_{inc}}{2^b} + \frac{\ell^4}{2^{2b}}\right).$$

Proof (Sketch) Consider all blocks x used as input to F during the course of the encryption oracle calls and incremental update oracle calls. Call all such blocks *used*. Note that a ciphertext block received by the adversary is computed as $C_i = F_{sk}(x_i) \oplus S_i$; in this case we say that x_i is *used with* S_i. First consider the case where the adversary outputs two documents, each containing ℓ blocks. Let $\mathcal{B} = \{L_1 \oplus S_0^\ell, R_1, \ldots L_n \oplus S_0^\ell, R_n\}$ be the set of blocks used as input to F during encryption of the chosen document. The adversary has non-zero advantage only in the following cases: (1) There exist $B_1, B_2, B_3, B_4 \in \mathcal{B}$ such that $B_1 = B_2$ and $B_3 = B_4$; (2) for some i, $L_i \oplus S_0^\ell$ is *used with* S_i^ℓ and R_i is *used with* S_i^r. This is similar to the analysis in Theorem 1, except that the pairwise-independent output whitening modifies conditions (1) and (2). The probability of (1) is $\mathcal{O}(\ell^4/2^{2b})$, and the probability of (2) is bound by $\mathcal{O}(q_e/2^b)$.

When the adversary outputs two modifications, the modification must (by definition) be of type replace. In this case, a random block L_i is chosen. The adversary's success will be 0 unless $L_i \oplus S_0^\ell$ is *used with* S_i^ℓ and R_i is *used with* S_i^r. The probability of this occurring is bounded by $\mathcal{O}(q_{inc}/2^b)$. ■

Theorem 4. *The inc-IAPM mode of encryption over modification space* $\mathcal{M} = \{\mathsf{replace}\}$ *is* $(t, q_e, \mu, q_{inc}; \epsilon)$*-secure in the sense of unforgeability, where:*

$$\epsilon = \frac{1}{2^b} + \mathcal{O}\left(\frac{q_e^2 + q_{inc}^2}{2^b}\right).$$

Proof (Sketch) Assume that seeds used in encryption oracle calls never repeat; such repetitions occur only with probability $\mathcal{O}(q_e^2/2^b)$. Let $\mathcal{C}$ be the set of ciphertexts received by the adversary from all its oracle calls, and let C' be the new ciphertext output by the adversary. Clearly, the adversary's success will be bounded by 2^{-3b} if C' uses a seed which has never appeared in any of the ciphertexts in $\mathcal{C}$. Now, assume the adversary uses a seed K which was used previously. Denote by $\mathcal{C}_K \subset \mathcal{C}$ the set of all ciphertexts which use this seed. Note that since seeds do not repeat for encryption oracle calls and we are dealing with replace operations only, all ciphertexts in $\mathcal{C}_K$ have the same length. By a similar argument to that of [17], the adversary has advantage at most $1/2^b$ if the length of C' is not equal to this length.

For each position i $(0 \le i \le n+1)$, let $\mathcal{C}_i^\ell$ be the set of ciphertext blocks corresponding to position C_i^ℓ in all ciphertexts in $\mathcal{C}_K$. Define $\mathcal{C}_i^r$ and $\mathcal{C}^*$ analogously. The adversary has success bounded by 2^{-b} if, for any i, $C_i'^\ell \notin \mathcal{C}_i^\ell$ (with similar statements holding for $C_i'^r$ and C'^*). This holds by pairwise independence of

the values in the output whitening sequence (and complete independence from sequences generated by other seeds).

Denote by $\mathcal{L}_i$ the set of values corresponding to decryption of blocks in $\mathcal{C}_i^\ell$, and by L'_i the value corresponding to decryption of block $C_i'^\ell$ in the adversary's output ciphertext. From the argument above, we may assume $L'_i \in \mathcal{L}_i$. Note that the adversary will have success probability bounded by 2^{-b} unless it is that case that $L_0, \ldots, L_{n+1}$ were all used *together* in some ciphertext in $\mathcal{C}_K$. This is true since the values $\cup_i \mathcal{L}_i$ are completely independent except for the relation (defined by the checksum) which holds between L_i's used in the same ciphertext. In other words, the adversary's success is bounded by 2^{-b} unless the blocks C_i^ℓ were all used together in some ciphertext in $\mathcal{C}_K$. A similar argument holds for the R_i (and hence the C_i^r). The same argument also holds for L_0, R_0, and D^*, and hence for C_0^ℓ, C_0^r, and C^*. Thus, unless L_0 has repeated for some pair of ciphertexts in $\mathcal{C}_K$ (which happens only with probability $\mathcal{O}(q_{inc}^2/2^b)$), the adversary has advantage at most 2^{-b} if it outputs a ciphertext different from those in $\mathcal{C}_K$. ■

4.3 RPC Mode

Theorem 5. $\mathrm{RPC}^{b,r}$ *mode over modification space* $\mathcal{M} = \{\mathsf{replace}, \mathsf{delete}, \mathsf{insert}\}$ *is* $(t, q_e, \mu, q_{inc}, \ell; \epsilon)$*-secure in the sense of indistinguishability, where:*

$$\epsilon = \mathcal{O}\left(\frac{\ell^2 + \ell\mu}{2^{2r}} + \frac{q_{inc}}{2^r}\right).$$

Proof (Sketch) Consider all blocks used as input to F during the encryption and incremental update oracle calls. Call all such blocks *used*. We first consider the case when the adversary outputs two documents, each containing ℓ blocks. Let $\mathcal{B} = \{r_1|\sigma_1|r_2, r_2|\sigma_2|r_3, \ldots, r_\ell|\sigma_\ell|r_0\}$ be the set of blocks used as input to F during the encryption of the chosen document. The adversary has non-zero advantage in only the following cases: (1) For some $B_1, B_2 \in \mathcal{B}$, it is the case that $B_1 = B_2$; (2) some $B \in \mathcal{B}$ is *used*. The probability of (1) is $\mathcal{O}(\ell^2/2^{2r})$, while the probability of (2) is $\mathcal{O}(\ell\mu/2^{2r})$.

When the adversary outputs two modifications, the adversary has a non-zero advantage only if the modifications were of type $\mathsf{replace}$ or insert. In either case, a random r' is chosen. The adversary has non-zero advantage only when $r'|\sigma|r_{i+1}$ is *used*. But the probability of this is bounded by $\mathcal{O}(q_{inc}/2^r)$. ■

Theorem 6. $\mathrm{RPC}^{b,r}$ *mode over modification space* $\mathcal{M} = \{\mathsf{replace}, \mathsf{delete}, \mathsf{insert}\}$ *is* $(t, q_e, \mu, q_{inc}; \epsilon)$*-secure in the sense of unforgeability, where:*

$$\epsilon = \frac{1}{2^{2r}} + \mathcal{O}\left(\frac{(q_e + q_{inc})^2}{2^{2r}}\right).$$

Proof (Sketch) Denote by $\mathcal{C}$ the set of all ciphertexts received by the adversary from its oracles. Note that the initial blocks of all ciphertexts in $\mathcal{C}$ are distinct,

Table 1. Comparison of the various modes. See text for details.

	Block size	Expansion	Cipher evals	IND	Unf	Comments
rECB-XOR	128	4	$6n$	$\frac{\mu\ell}{2^{128}}$	$\frac{\mu^2}{2^{64}}$	
	256	2.7	$4.6n$	$\frac{\mu\ell}{2^{256}}$	$\frac{\mu^2}{2^{64}}$	
inc-IAPM	128	2	$2n$	$\frac{q_{tot}}{2^{128}}$	$\frac{q_{tot}^2}{2^{128}}$	only replace
RPC	128	4	$4n$	$\frac{q_{inc}}{2^{48}}$	$\frac{q_{tot}^2}{2^{96}}$	
	256	2	$2n$	$\frac{q_{inc}}{2^{64}}$	$\frac{q_{tot}^2}{2^{128}}$	

except with probability $\mathcal{O}((q_e + q_{inc})^2/2^{2r})$. Now, assume they are all distinct. Let the ciphertext output by the adversary be denoted C'.

Denote by $\mathcal{C}_0$ the set of values for the first block, for all ciphertexts in $\mathcal{C}$. Note that the adversary's success probability is bounded by 2^{-2r} if $C'_0 \notin \mathcal{C}_0$. Let $C \in \mathcal{C}$ be such that $C_0 = C'_0$ (by the assumption above, this defines a unique C). C and C' either consist of the same blocks (possibly in different order), or there exists a block which appears a different number of times in C and C'. In the first case, note that if the blocks are in different order, then the adversary's probability of success is bounded by 2^{-2r}. In the second case, since the random nonces used for the additional block are independent of all other nonces in the sequence (in particular, the value r_1 defined by block C_0), the adversary's probability of success is bounded by 2^{-2r}. ■

5 A Comparison of the Various Schemes

The encrypt-then-incremental-MAC approach is appealing because it is so simple. Unfortunately, incremental MAC algorithms are, in practice, difficult to design. The constructions of [3,6] are number-theoretic in nature and involve algebraic operations over groups instead of fast bit-wise operations such as XOR. A tree-based suggestion of [4] is practical, but update operations require $O(\log |D|)$ block cipher evaluations. The only truly practical incremental MAC (with constant update time) of which we are aware is the XOR-scheme of [4]. We call the resulting scheme (cf. Section 3.1) rECB-XOR.

Consider implementation with a 128-bit block cipher, and using $r = 64$ bits of padding for the MAC. This results in ciphertext which is more than four times the length of the plaintext (for an n-block document, rECB encryption results in $2n$ blocks, and the tag computed by the MAC is an additional $2n$ blocks). Encryption of a document from scratch is computationally expensive. An n-block document requires $2n$ block cipher evaluations for the encryption and an additional $4n$ block cipher evaluations to compute the MAC, for a total of $6n$ block cipher evaluations. Furthermore, the security guarantee (for unforgeability) is not as strong as one might like, since it is limited by the factor of $\mu^2/2^{64}$ even though the underlying block cipher is 128 bits long. On the other hand, the scheme

supports incremental insert, delete, and replace operations, and each can be done in constant time. Performance of the mode is enhanced slightly when using a 256-bit block cipher. In this case, keeping $r = 64$ results in ciphertext only 2.7 times the length of the plaintext and only $4.6n$ block cipher evaluations are required for encryption of an n-block document. The security guarantee (for unforgeability) is still relatively poor.

The inc-IAPM mode results in ciphertext which is twice the length of the plaintext. Encryption of an n-block document requires only $2n + \mathcal{O}(1)$ block cipher evaluations (this is comparable to the gain in efficiency of IAPM vs. standard encrypt-then-MAC [17]). The incremental replace operation can be done using only a constant[6] number of block cipher evaluations independent of the document size. We note that this mode is efficient without requiring the larger, 256-bit blocksize. Finally, the security guarantees for inc-IAPM are stronger than that for the previous scheme, most strikingly in the case of unforgeability.

The biggest drawback of the inc-IAPM mode is that it does not support incremental delete and insert operations. This might be acceptable in some contexts; for example, when maintaining a database with a fixed number of entries. Since the number of entries does not change, delete and insert operations are not required. However, it is still important to support incremental replace operations, since this is what will be required when entries are modified.

RPC mode supports incremental replace, insert, and delete modifications in constant time. The ciphertext expansion and computational efficiency, however, are not as good as the inc-IAPM mode. When using a 128-bit block cipher, one might set $r = 48$ which results in ciphertext expansion by a factor of 4 and requires $4n$ block cipher evaluations for encryption of an n-block document. The security (especially in the sense of indistinguishability) is poor, being limited by $q_{inc}/2^{48}$. However, this mode performs much better when using a 256-bit block cipher. Here, we may set $r = 64$, resulting in ciphertext expansion of only 2 and requiring only $2n$ block cipher evaluations for encrypting an n-block document. The security may now be acceptable for many applications.

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 '97.
3. M. Bellare, O. Goldreich, and S. Goldwasser. Incremental Cryptography: The Case of Hashing and Signing. Crypto '94.
4. M. Bellare, O. Goldreich, and S. Goldwasser. Incremental Cryptography and Application to Virus Protection. STOC '95.

[6] Technically speaking, the replace operation may require a polylogarithmic number of *bit-wise* operations to compute members of the whitening sequence. However, these are bit-wise operations, which are about two orders of magnitude faster than block cipher evaluations.

5. M. Bellare, J. Kilian, and P. Rogaway. On the Security of Cipher Block Chaining. CRYPTO '94.
6. M. Bellare and D. Micciancio. A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. Eurocrypt '97.
7. M. Bellare and C. Namprempre. Authenticated Encryption: Relations Among Notions and Analysis of the Generic Composition Paradigm. Asiacrypt 2000.
8. M. Bellare and P. Rogaway. Encode-then-Encipher Encryption: How to Exploit Nonces or Redundancy in Plaintexts for Efficient Cryptography. Asiacrypt 2000.
9. M. Blaze. A Cryptographic File System for Unix. 1st ACM Conference on Computer and Communications Security, 1993.
10. M. Fischlin. Incremental Cryptography and Memory Checkers. Eurocrypt '97.
11. M. Fischlin. Lower Bounds for the Signature Size of Incremental Schemes. FOCS '97.
12. V. Gligor and P. Donescu. Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes. FSE 2001.
13. O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. JACM 33(4): 792–807 (1986).
14. S. Goldwasser and S. Micali. Probabilistic Encryption. JCSS, 28: 270-299, 1984.
15. S. Halevi. An Observation Regarding Jutla's Modes of Operation. Available at `http://eprint.iacr.org/2001/015`.
16. ISO 8372, "Information Processing—Modes of Operation for a 64-bit Block Cipher Algorithm," International Organization for Standardization, Geneva, Switzerland, 1987.
17. C. S. Jutla. Encryption Modes with Almost-Free Message Integrity. Eurocrypt 2001, to appear. Also available at `http://eprint.iacr.org`.
18. J. Katz and M. Yung. Complete Characterization of Security Notions for Probabilistic Private-Key Encryption. STOC 2000.
19. J. Katz and M. Yung. Unforgeability and Chosen-Ciphertext-Secure Modes of Operation. FSE 2000.
20. M. Luby. *Pseudorandomness and Cryptographic Applications*, Chapter 14. Princeton University Press, 1996.
21. D. Micciancio. Oblivious Data Structures: Applications to Cryptography. STOC '97.
22. See: `http://www.nist.gov/modes`.
23. National Bureau of Standards, NBS FIPS PUB 81, "DES Modes of Operation," U.S. Department of Commerce, 1980.
24. P. Rogaway. OCB Mode: Parallelizable Authenticated Encryption. Available at [22].

ZIP Attacks with Reduced Known Plaintext

Michael Stay

AccessData Corporation
2500 N. University Ave. Ste. 200
Provo, UT 84606
staym@accessdata.com

Abstract. Biham and Kocher demonstrated that the PKZIP stream cipher was weak and presented an attack requiring thirteen bytes of plaintext. The *deflate* algorithm "zippers" now use to compress the plaintext before encryption makes it difficult to get known plaintext. We consider the problem of reducing the amount of known plaintext by finding other ways to filter key guesses. In most cases we can reduce the amount of known plaintext from the archived file to two or three bytes, depending on the zipper used and the number of files in the archive. For the most popular zippers on the Internet, there is a fast attack that does not require any information about the files in the archive; instead, it gets doubly-encrypted plaintext by exploiting a weakness in the pseudorandom-number generator.

1 Introduction

PKZIP is a compression / archival program created by Phil Katz. Katz had the foresight to document his file format completely in the file APPNOTE.TXT, distributed with every copy of PKZIP; there are now literally hundreds of "zipper" programs available, and the ZIP file format has become a *de facto* standard on the Internet.

In [BK94] Biham and Kocher demonstrated that the PKZIP stream cipher was weak and presented an attack requiring thirteen bytes of plaintext. Eight bytes of the plaintext must be contiguous, and all of the bytes must be the text that was encrypted, which is usually compressed data. [K92] shows that the compression method used at the time, *implode*, produces many predictable bytes suitable for mounting the attack.

Most zippers available today implement only one of the compression methods defined in APPNOTE.TXT, called *deflate*. *Deflate* uses Huffman coding followed by a variant of Lempel-Ziv. Once the dictionary reaches a certain size, the process starts over. Since the Huffman codes for any of the data depend on a great deal of surrounding data, one is forced to guess the plaintext unless one has the original data. The difficulty of getting known plaintext was one reason Phil Zimmerman decided to use *deflate* in PGP [PGP98]. Practically speaking, if one has enough of the original file to get the thirteen bytes of plaintext required for the attack in [BK94], one has enough to break the encryption almost instantly.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 125–134, 2002.

Without the original file, all is not lost; we have the file's type as indicated by its extension, and we have its size. The ZIP file format requires at least one byte of known plaintext for filtering incorrect passwords. Most zippers also encrypt output from a pseudorandom number generator that is vulnerable to attack.

It is the author's opinion that the only reason the PKZIP cipher has held up so well in light of [BK94] is the high entropy of the data produced by the *deflate* algorithm and the related difficulty of getting enough plaintext. This paper treats the question of how far we can reduce the plaintext requirement and still break the cipher with a practical amount of work.

1.1 The PKZIP Stream Cipher

The PKZIP stream cipher was designed by Roger Schaffely and is fully described in the file APPNOTE.TXT found in most PKZIP distributions. The internal state of the cipher consists of three 32-bit words: *key*0, *key*1, and *key*2. These values are initialized to 0x12345678, 0x23456789, and 0x34567890, respectively. The internal state is updated by mixing in the next plaintext byte. The first and third words are updated using the linear feedback shift register known as CRC-32; the second word is updated using a truncated linear congruential generator. The output byte is the result of a truncated pseudo-squaring operation. (See Figure 1.)

```
unsigned char PKZIP_stream_byte (unsigned char pt)
{
   unsigned short temp;
   key0 = crc32 ( key0, pt );
   key1 = ( key1 + LSB( key0 ) ) * 0x08088405 + 1;
   key2 = crc32 ( key2, MSB( key1 ) );
   temp = key2 | 3;
   return LSB( ( temp * (temp ^ 1) ) >> 8);
}
```

where 'unsigned char' is an 8-bit integer; 'unsigned short' is a 16-bit integer; | is bitwise OR; ^ is XOR; >> is right shift; 'LSB' and 'MSB' are the least-significant and most-significant bytes, respectively; and '0x' is a prefix indicating hexadecimal.

For the purposes of this paper, we define *crc*32() to be

```
unsigned long crc32( unsigned long crc, unsigned char b)
{
   return ( (crc >> 8) ^ crctab [ LSB( crc ) ^ b ] );
}
```

where 'unsigned long' is a 32-bit integer.

The old LFSR state is shifted right eight bits and XORed with the 32-bit entry of a byte-indexed table to produce the new state. The index is the low byte of the old state XORed with b. The function is linear; that is,

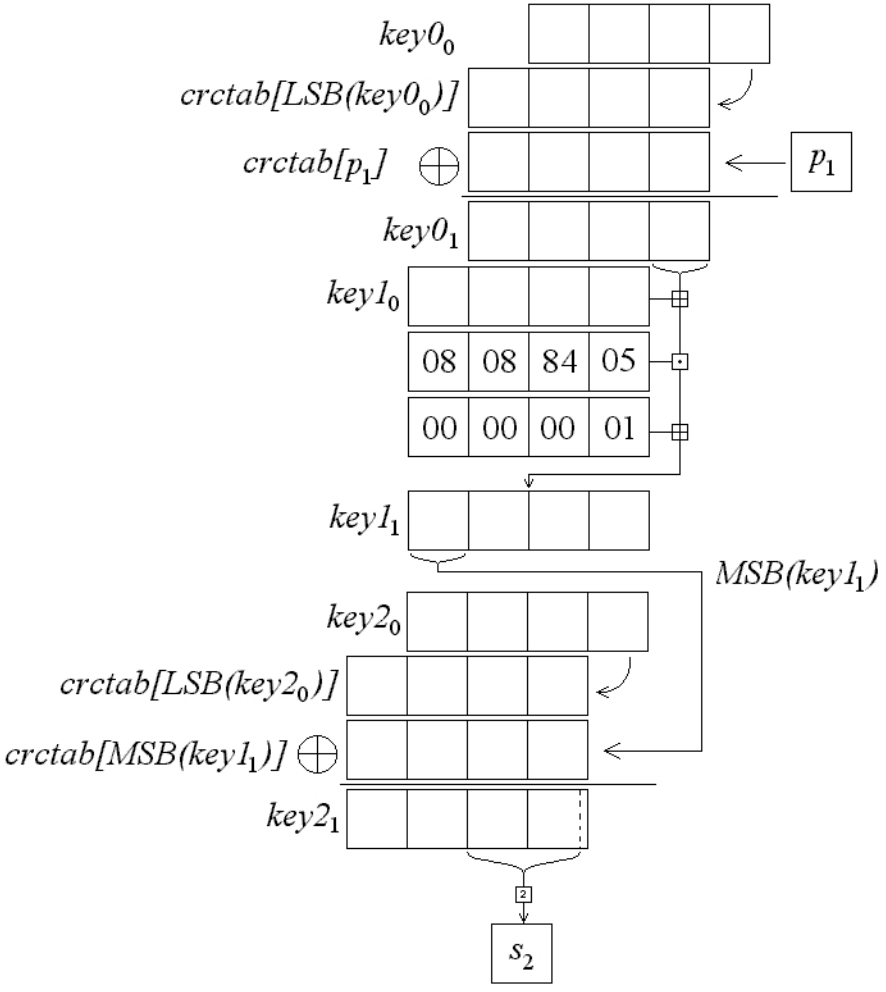


Fig. 1. The PKZIP stream cipher. CRC-32, TLCG, CRC-32, truncated pseudo-square.

```
crctab[ x ^ y ] = crctab[ x ] ^ crctab[ y ].
```

The cipher is keyed by encrypting the user's password and throwing away the corresponding stream bytes. The stream bytes produced after this point are XORed with the plaintext to produce the ciphertext.

The crux of all of our attacks is the fact that there is almost no diffusion in the internal state. Of the ninety-six bits of internal state, eight bits of *key*0 affect *key*1; eight bits of *key*1 affect *key*2; and fourteen bits of *key*2 affect the output stream byte.

1.2 Encrypted File Format

Zippers must prepend twelve bytes to the beginning of the file to be encrypted. The ZIP file format specifies that the first eleven should be random and that the last should be the low byte of the archived file's CRC. The entire CRC is stored in plaintext, and this byte serves as a password filter. Some zippers, like InfoZIP [IZ] and WinZip [WZ], store ten random bytes and the low two bytes of the CRC.

We assume that *deflate* was the algorithm used to compress the underlying data. The author did a crude statistical test on a few hundred files of varying types and sizes and found that given the file type, as indicated by its extension, and its size, one can guess about the first two and a half bytes of the compressed file. Since the checksum bytes are at the very end of the prepended header, we can use them to augment the plaintext from the file in mounting our attacks.

2 Biham and Kocher's Attack

For completeness, we review [BK94]'s results. We begin with some terminology. Bits are numbered from right to left: bit 0 is the ones' place, bit 1 is the twos' place, bit 2 is the fours' place, etc. Let p_i be the ith known-plaintext byte, $i = 1$, 2, 3, ... Let s_i be the ith stream byte. Let $key0_i$, $key1_i$, and $key2_i$ be the value of $key0$, $key1$, and $key2$ after processing p_i. Note that s_1 is a function of the random header and the password; it is independent of the plaintext. In general, bits 2 through 15 of $key2_i$ determine s_{i+1}.

Their attack proceeds as follows:

- XOR ciphertext and known plaintext to get known stream bytes s_1 through s_{13}.
- Guess 22 bits of $key2_{13}$.
- This guess combined with s_{13} is enough to fill in eight more bits of $key2_{13}$, for a total of thirty. s_{12} provides enough information to derive 30 bits of $key2_{12}$ and the most significant byte of $key1_{13}$. In general, each stream byte s_i allows us to calculate thirty bits of $key2_{i-1}$ and the most significant byte of $key1_i$.
- We continue using stream bytes to make a list of the most significant bytes of $key1_{13}$ through $key1_8$.
- For each list, we find 2^{16} possibilities for the low 24 bits of $key1_{13}$ through $key1_9$ by calculating the low byte of $(key1_i + LSB(key0_{i+1}))$ such that we get the right high byte of $key1_{i+1}$.
- From each of the 2^{16} lists of complete $key1$'s, derive the low bytes of $key0_{13}$ through $key0_{10}$.
- Once we have the low bytes of $key0_{10}$, $key0_{11}$, $key0_{12}$, and $key0_{13}$, we can use our knowledge of the plaintext bytes to invert the CRC function, since it's linear, and find the complete internal state at one point along the encryption.
- Once we have the complete internal state, we can decrypt backwards as far as we want; we decrypt the ciphertext corresponding to p_1 through p_5 and filter out wrong keys.

We can break a file with work equivalent to encrypting around 2^{38} bytes and negligible memory. We need a total of thirteen bytes of known plaintext: eight for the attack, and five to filter the 2^{38} keys that remain. This is an upper bound; each additional byte of plaintext eliminates approximately one list (see [BK94], fig. 1).

2.1 Minor Improvement in the Amount of Plaintext Required

[BK94] throws away six bits in $key1_7$. By using them, we can reduce the plaintext requirement to twelve bytes at the cost of increasing the work factor by four.

2.2 More Files in the Archive

If we have more than one file in the archive, we can make the reasonable assumption that they were encrypted with the same password. Zippers encrypt at least one check byte into every encrypted file to verify that the user entered the correct password. Once we have the complete internal state of the cipher, we can run it backwards to the beginning of the file and read out $key0$, $key1$, and $key2$. Since this state is the same at the beginning of each file (it only depends on the password), we can decrypt the check byte in each file and use it to filter with instead of known plaintext from a single file. This also works if the files are in different archives, but have the same password.

If the file was created in a zipper with two checksum bytes, we can break the file with work equivalent to encrypting $11 * 2^{40} \approx 2^{43}$ bytes. We need two checksum bytes followed by only four more known plaintext bytes in one file, and three other files in the archive (six check bytes) to filter the 2^{40} possible keys. The factor of eleven in the estimate above is due to the fact that to decrypt the checksum byte, we must decrypt the first eleven bytes of the random header.

If there is only one checksum byte per archived file, we can break the cipher with the nearly the same amount of work, but we need seven files in the archive and five bytes of known plaintext in addition to the checksum byte in the first file.

3 Divide and Conquer

The limited diffusion of the internal state prompts us to ask how much of the state we need to guess to process one byte. If it is small enough, we can guess it and filter out keys that won't work with our known stream bytes, then proceed to the next part.

It turns out that we can get by with as few as 23 bits (See Figure 2.) Note that we don't need to guess 16 bits of $key0_0$ to calculate the low byte of $key0_1$: if we distribute the XOR in the definition of $crc32()$, we see that we only need to guess 8 bits of $crc32(key0_0, 0)$:

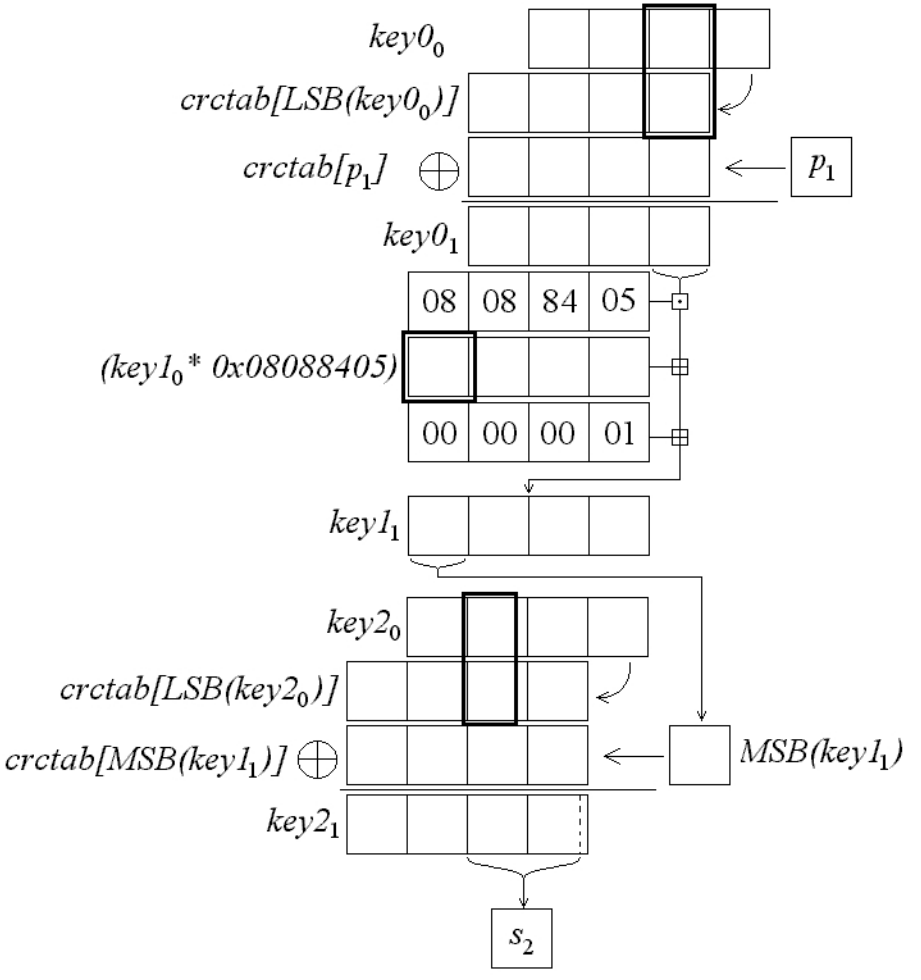


Fig. 2. The twenty-three bits involved in generating a stream byte.

```
LSB(key0_1) = LSB( crc32( key0_0, p_1 ) )
            = LSB( crc32( key0_0, 0 ) ) XOR LSB( crctab[ p_1 ] ).
```

Now we distribute the multiplication across the addition in the next step:

```
MSB(key1_1) =  MSB( ( key1_0 + LSB(key0_1) ) * 0x08088405 + 1 )
(A)         =  MSB( LSB( key0_1 ) * 0x08088405 ) +
(B)            MSB( key1_0 * 0x08088405 ) + possible carry bit.
```

We separate the equation into parts we know (A) and parts we need to guess (B), and find we need to guess nine bits, including a possible carry bit. Note that since we know the low bits of $(LSB(key0_1) * 0x08088405)$, the carry bit will usually give us more than one bit of information in the form of an upper or lower bound on the rest of $key1_0 * 0x08088405)$ that we haven't guessed yet.

Given a stream byte s_{i+1}, we can find sixty four values for bits 2..15 of $key2_i$. It's easy to see why: fourteen bits of $key2_i$ produce eight bits of s_{i+1}, so there are six left over. We can create a table of 256 x 64 bytes such that given s_{i+1} and bits 10..15 of $key2_i$, we can look up bits 2..9 of $key2_i$. We call this the preimage table.

We guess bits 10..15 of $crc32(key2_0, 0)$ and use s_2, the preimage table, and $crctab[MSB(key1_1)]$ to find bits 2..9 of $crc32(key2_0, 0)$. We end up with 2^{23} key guesses.

To find the next part of the internal state, we have to guess about the same amount. This guess is not illustrated, but we basically guess about eight more bits of information in each of the three keys. The only complicated part is separating what we know about $key1$ from what we don't.

We guess bits 8..15 of $crc32(key0_0, 0)$ directly; the next guess involving $key1$ is a little more complicated:

```
MSB(key1_2) =  MSB( (key1_1 + LSB(key0_2)) * 0x08088405 + 1 )

            =  MSB( LSB(key0_2) * 0x08088405 ) +
               MSB( key1_1 * 0x08088405 ) + possible carry bit

            =  MSB( LSB(key0_2) * 0x08088405 ) +
               MSB( (key1_0 + LSB(key0_1)) * 0xD4652819 ) +
               possible carry bit

(A)         =  MSB( LSB(key0_2) * 0x08088405 ) +
(A)            MSB( LSB(key0_1) * 0xD4652819 ) +
(B)            MSB( key1_0 * 0xD4652819 ) + possible carry bit.
```

Again, (A) is known and we have to guess (B) nine bits, including a possible carry bit. The carry bit establishes an upper or lower bound on $(key1_0 * 0xD4652819)$. We end this filter by guessing bits 16..23 and bits 0..1 of $crc32(key2_0, 0)$ and calculating a stream byte. We have guessed 27 more bits, but the output byte has to match s_3, so we expect $2^{23+27-8} = 2^{42}$ key guesses to pass this filter.

At this point, we have guessed 24 bits of $crc32(key2_0, 0)$ and we know s_1. From this we can calculate, on average, one full value of $key2_0$. There are also only around 2^{13} possibilities for $key1_0$ due to the restrictions from the carry bits. So the third stage consists of guessing bits 16..23 of $crc32(key0_0, 0)$ and running through the 2^{13} possible values for $key1_0$. We expect $2^{42+13+8-8} = 2^{55}$ key pieces to pass this filter.

Finally, we guess the last eight bits of $key0_0$ and we have a complete internal state. We will have 2^{63} complete keys to filter with other bytes, whether they

are in the archived file or in checksum bytes in other files. The cost is approximately the same as encrypting 2^{63} bytes under the stream cipher. The plaintext requirement is four bytes total; at least one of these may come from the file's own check byte(s).

This is 128 times faster than guessing three stream bytes and using [BK94].

4 Parallel Divide and Conquer Attack

InfoZIP is a cross-platform freeware zipper distribution. Because the C source code is readily available and is free, it forms the basis of most non-PKZIP zippers, including the very popular WinZip and NetZip. According to CNET's Download.com [DL], WinZip and NetZip constitute over 96% of the total archiver downloads.

APPNOTE.TXT does not specify how to generate the prepended random bytes; it only says that they are used to scramble the internal state of the cipher and are discarded after decryption. InfoZIP implements it as follows:

1. `srand( time(NULL) ^ getpid() )`
2. For each file in the order they are stored,
3. Generate ten random bytes by calling *rand()* ten times and discarding all but the high eight bits of each return value.
4. Initialize the cipher with the password.
5. Encrypt the ten random bytes.
6. Append the low two bytes of the checksum.
7. Reinitialize the cipher with the password.
8. Encrypt the twelve-byte header and the compressed file.

Note that the random bytes were encrypted *twice*: once in step 5, and again in step 8.

rand() is usually implemented as a truncated linear congruential generator. WinZip and NetZip use Microsoft Visual C++'s implementation, which has a 31-bit seed:

```
unsigned long seed;
void srand( unsigned long s ) { seed = s; }
unsigned short rand()
{
   seed = 0x343FD * seed + 0x269EC3;
   return ( ( seed >> 16) & 0x7FFF );
}
```

Let $r_{i,j}$ be the jth random byte in the ith archived file; $i, j = 1, 2, 3, ...$ Note that the internal state of the cipher is the same both times $r_{i,1}$ is encrypted. Since XOR is its own inverse, $r_{i,1}$ is *decrypted* for all i. Also, every $r_{i,1}$ reveals the high eight bits of the internal state of the random number generator.

Since *rand*() is linear, we can compute two new constants for a generator such that it outputs every tenth output of the original. We know the upper eight bits of the generator, so we guess the low 23 bits and start generating every tenth output and comparing them to the revealed bytes. Five archived files suffice to

determine uniquely the seed that was used in the random number generator, and therefore every $r_{i,j}$.

Let us emphasize that we do *not* have known plaintext at this point, in the sense that [BK94] requires. The random bytes were encrypted twice, so we do not know the actual output of the stream cipher during the first and second encryption. What we *can* derive is the XOR of these stream bytes.

4.1 The Attack

We can adapt the divide-and-conquer algorithm from section 3 to use this information. Once we know the "random" headers, we can exploit the fact that the internal state was the same at the beginning of each embedded file and filter guesses with multiple known plaintext bytes in parallel, instead of being restricted to one byte as in section 3.

Let $s_{i,j,k}$ be the jth stream byte of the kth encryption of the bytes in the ith archived file; i, $j = 1, 2, 3, ...$; $k = 1, 2$. We guess the same 23 bits as in section 3, but since we don't know the actual value of $s_{1,2,1}$, we have to guess it, too. It is equivalent, and more convenient, to guess bits 2..9 of $crc32(key2_0, 0)$. Now we have a prediction for $s_{1,2,1}$, and can derive $s_{1,2,2}$. We don't have any information at all about $s_{i,1,1}$, since it's the same as $s_{i,1,2}$ and cancels out. We guess it, and check to see that the second encryption spits out $s_{1,2,2}$. We have to guess a carry bit for the second encryption, too, so of the $2^{23+8+8+1} = 2^{40}$ key guesses, we expect $2^{40-8} = 2^{32}$ key pieces to pass this filter on this file.

We want to filter out all but the correct guess at this stage; fortunately, we know that the state we are trying to guess was the same at the start of each encryption. We have an eight-bit value to filter with in each file,
$s_{i,2,1}$ XOR $s_{i,2,2}$, but we also guess two carry bits, so with four more files in the archive, we can reduce the number of false positives to around $2^{32-6*4} = 256$. Note that we now have *ten* carry bits putting restrictions on $key1_0$ instead of just one.

We continue to the next byte of each file. This time we guess the same 26 bits as in section 3 plus two carry bits, one for each encryption. With five files, we have 30 bits to filter with. We expect that $2^{8+26+2-30} = 2^6 = 64$ key guesses survive the second stage. Total work for this stage is $2^{8+26+2} = 2^{36}$ byte encryptions.

At this point we can derive $key2_0$ as before. Due to all the carry bit restrictions, we only have on the order of 2^8 possible $key1_0$'s. We guess eight more bits of $crc32(key1_0, 0)$ and run through all the remaining $key1_0$'s. Since we aren't guessing carry bits any more, we have 40 bits to filter with. $2^{6+16-40} < 1$, and we expect that only the correct guess survives. Finally, we guess the last eight bits of $crc32(key1_0, 0)$ and only the correct guess survives.

Experimentally, we have found that a key guess passes the second stage only if our guess for $s_{i,1,1}$ is correct. This usually occurs about one quarter of the way through the first 40-bit keyspace. After that, we only try one value for $s_{i,1,1}$ instead of 256 and the rest of the attack takes at most a few minutes.

The work done in the first stage dwarfs the rest of the work needed. The total work is therefore about the same as encrypting 2^{39} bytes. We assume that

there are five files in the archive that were encrypted consecutively as described above. Decrypting a file created with this kind of weak PRNG usually takes under two hours on a 500 MHz Pentium II. One can then take the three keys and use [BK94]'s second algorithm to derive a password, if one desires, although the three keys suffice to decrypt the files.

Table 1. PKZip Attack Complexity. Files are assumed to have been archived with two checksum bytes.

Attack	Archived files	Plaintext bytes	Complexity
BK94	1	13	2^{38}
BK94 (tradeoff)	1	12	2^{40}
BK94 (tradeoff)	4	6	$11 * 2^{40}$
Divide and conquer	1	2	2^{63}
Parallel divide and conquer (WinZip)	5	0	2^{39}

5 Conclusion

The PKZIP stream cipher is very weak. The *deflate* algorithm makes it harder to get plaintext, but in most cases we can reduce the plaintext requirement to the point where one can guess enough plaintext based on file type and size alone. The most popular zippers on the internet are also susceptible to an attack that runs in two hours on a single PC based on known plaintext provided by the application and independent of the archived files themselves.

References

[BK94] Biham, Eli and Paul Kocher. "A Known Plaintext Attack on the PKZIP Stream Cipher." Fast Software Encryption 2, Proceedings of the Leuven Workshop, LNCS 1008, December 1994.

[DL] `http://download.cnet.com/downloads/0,10151,0-10097-106-0-1-5,00.html?tag=st.dl.10097_106_1.lst.lst&`

[IZ] `ftp://ftp.freesoftware.com/pub/infozip/`

[K92] Kocher, Paul. *ZIPCRACK 2.00 Documentation.* 1992. `http://www.bokler.com/bokler/zipcrack.txt`

[PKZ] `http://www.pkware.com`

[PGP98] *User's Guide, Version 6.0.* Network Associates, Inc., 1998. p.145. `http://www.nai.com`

[WZ] `http://www.winzip.com`

Cryptanalysis of the SEAL 3.0 Pseudorandom Function Family

Scott R. Fluhrer

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

Abstract. We present an attack on the SEAL Pseudorandom Function Family that is able to efficiently distinguish it from a truly random function with 2^{43} bytes output. While this is not a practical attack on any use of SEAL, it does demonstrate that SEAL does not achieve its design goals.

1 Introduction

SEAL is a series of encryption algorithms designed by Phillip Rogaway and Don Coppersmith. The most recent version is SEAL 3.0, which is described in [6]. In this paper, all references to SEAL without an explicit version number are to version 3.0.

SEAL is designed to be a cryptographic object called a pseudorandom function family, which were defined in [2]. This is an object that, under the control of a key, maps a fixed length input string to an output string in a manner that appears random. SEAL, in particular, has a 160 bit key, and maps a 32 bit input into a 64kbyte output.

Rogaway and Coppersmith gave as an explicit design goal that it would be computationally infeasible to distinguish (with probability significantly greater than 0.5) SEAL with a random fixed key from a truly random function. This paper shows a truncated differential attack that is able to distinguish it, given 2^{43} bytes of output and $O(2^{43})$ work. We note that, since SEAL with a fixed key defines only 2^{48} bytes of output, this attack requires relatively close to the total amount of data that an attacker can conceivably examine with a single key.

This paper is structured as follows. In Section 2, the SEAL pseudorandom function family is described, and previous analysis are summarized. Section 3 lists some key observations about the internals of the SEAL update function, and sections 4 and 5 provide the actual attack. Section 6 presents results of this attack on weakened variants of SEAL, and section 7 discusses what aspects of SEAL allowed this attack to be successful, and summarizes our conclusions.

2 Description of SEAL and Other Work

SEAL is a length increasing pseudorandom function family that, under the control of a 160-bit key, expands a 32-bit string into a 2^{19}-bit string. Internally,

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 135–143, 2002.

SEAL expands the 160-bit keys into 3 secret tables R, S, T, which contain respectively 256, 256, and 512 32-bit values. These tables are fixed once the key has been defined.

To generate 2^{13} bits of output, SEAL takes the 32-bit input string n and 6 bits of submessage index[1] ℓ and expands it into 256 bits of state (the 32 bit A, B, C, D variables and the 32 bit n_1, n_2, n_3, n_4 variables using the R and T tables). Then, SEAL steps through a 6 bit block index[2] i, and alternates between two slightly different noninvertible update functions to update A, B, C, D using the T table. At one point during the update, outputs A, B, C, D combined with the 4 32-bit elements from the S table indexed by i. SEAL iterates through this 64 times to generate 2^{13} bits. To generate the full 2^{19} bits, SEAL steps through all 2^6 submessage indices, and concatenates the results.

For the reader's convenience, the pseudocode for the update function is copied from [6] and is listed as Table 1. In this code, α denotes the key (which is implicitly used to derive the secret R, S, T tables), n is the 32 bit input string, L is the number of output bits desired, λ denotes the empty string, $Initialize$ is a function that sets A, B, C, D, n_1, n_2, n_3, n_4 to key, message, and submessage index dependent 32-bit values, $\&$ and $\oplus$ denote bit-wise *and* and $exclusive-or$, $+$ denotes addition modulo 2^{32}, $\rangle\rangle\rangle$ denotes a right circular shift, and $\|$ denotes concatenation.

This attack ignores the key expansion and the $Initialize$ function, idealizing them as truly random processes, and so the details of how they are specified are not listed in this paper.

SEAL was originally published as version 1.0 in [5]. Version 2.0, modified to use the revised SHA-1 algorithm in the key expansion, appeared in [4]. Handschuh and Gilbert discovered an attack, described in [3], that is able to distinguish both SEAL 1.0 and SEAL 2.0 from a random function, and was also able to gain some information about the hidden table entries. In response to this attack, Rogaway and Coppersmith revised it in [6] to SEAL version 3.0, which is the subject of this paper. There are no other attacks described in the open literature.

3 Basics of P/Q Collisions

This attack is based on a specific truncated differential within the update function. This differential is between two executions of steps 1 through 8 of the next state function, and is defined to be when all of the following occur:

[1] The submessage index is defined to be the value indicating which 1kbyte section of the 64kbyte output SEAL is currently generating on behalf of a particular input string. This terminology is specific to this paper, and while it is represented by the variable ℓ in Figure 2 of [6], it is not given an explicit name.

[2] The block index is defined to be the value indicating which 16 byte section of the 1kbyte submessage SEAL is currently generating. This terminology is specific to this paper, and while it is represented by the variable i in Figure 2 of [6], it is not given an explicit name.

Table 1. Cipher mapping 32-bit position index n to L-bit string $SEAL(\alpha, n, L)$ under the control of α-derived tables T, R, S

function SEAL(α, n, L)
$y = \lambda$;
for $\ell \leftarrow 0$ **to** ∞ **do**
Initialize(n,ℓ, A,B,C,D, n_1,n_2,n_3,n_4);
for $i \leftarrow 1$ **to** 64 **do**

1 $P \leftarrow A$&0x7fc; $B \leftarrow B + T[P/4]; A \leftarrow A)\rangle\rangle 9; B \leftarrow B \oplus A;$
2 $Q \leftarrow B$&0x7fc; $C \leftarrow C \oplus T[Q/4]; B \leftarrow B)\rangle\rangle 9; C \leftarrow C + B;$
3 $P \leftarrow (P + C)$&0x7fc; $D \leftarrow D + T[P/4]; C \leftarrow C)\rangle\rangle 9; D \leftarrow D \oplus C;$
4 $Q \leftarrow (Q + D)$&0x7fc; $A \leftarrow A \oplus T[Q/4]; D \leftarrow D)\rangle\rangle 9; A \leftarrow A + D;$
5 $P \leftarrow (P + A)$&0x7fc; $B \leftarrow B \oplus T[P/4]; A \leftarrow A)\rangle\rangle 9;$
6 $Q \leftarrow (Q + B)$&0x7fc; $C \leftarrow C + T[Q/4]; B \leftarrow B)\rangle\rangle 9;$
7 $P \leftarrow (P + C)$&0x7fc; $D \leftarrow D \oplus T[P/4]; C \leftarrow C)\rangle\rangle 9;$
8 $Q \leftarrow (Q + D)$&0x7fc; $A \leftarrow A + T[Q/4]; D \leftarrow D)\rangle\rangle 9;$
9 $y \leftarrow y \parallel B + S[4i-4] \parallel C \oplus S[4i-3] \parallel D + S[4i-2] \parallel A \oplus S[4i-1];$
10 **if** $|y| \geq L$ **then return** $(y_0 y_1 \ldots y_{L-1})$;
11 **if** $odd(i)$ **then** $(A, B, C, D) \leftarrow (A + n_1, B + n_2, C \oplus n_1, D \oplus n_2)$
else $(A, B, C, D) \leftarrow (A + n_3, B + n_4, C \oplus n_3, D \oplus n_4)$

1. The P and Q internal variables have zero differential at all points through steps 1 through 8.
2. Both sides of the differential have the same block index (i.e. i has zero differential).
3. The common block index is not the first (i.e. $i \neq 1$).

We will call an occurrence of this differential a *P/Q collision*. We will see that an occurrence of a P/Q collision can be seen in the differential output states both before and after the update, and will use that to provide the distinguisher.

We will use the syntax x_n to indicate the differential in variable X at the start of the execution of line[3] n. We will also use the syntax x_{output} to represent the differential in the output in line 9 that the variable X contributes to (including the contribution of the S table). In addition, we will also use the convention that bit 0 is the least significant bit, and bit 31 is the most significant in a 32 bit word.

Consider two separate iterations of lines 1-8 of the update function where a P/Q collision occurs. Then, because we know that p_2 has zero differential (by the definition of a P/Q collision) we know that bits 10-2 of a_1 must have zero differential, which implies that bits 31-25 and 1-0 of a_2 must have zero differential. And, we know, since both p_5 and p_6 have zero differential, that bits 10-2 of a_5 must also have zero differential. Since bits 10-2 of d_4 must have zero differential (because of step 4) and bits 31-25 and 1-0 of d_4 must have zero differential (because of step 8), then bits 10-0 of a_4 must have zero differential.

[3] All references to lines refer to the numbered lines of code in Table 1.

Then, along with step 2, we see that, because the carries into bit 2 of the sum $B+T[P/4]$ in line 1 may be different[4], there is an additive differential within the set {0, +1, -1} (with probabilities 1/2, 1/4, 1/4 respectively[5]. We will shorthand this relationship by saying that bits 10-2 of b_1 has a differential of ϵ. In addition, we will shorthand a differential that represents the sum of two such independent additive differentials (in other words, has an additive differential within the set {0, +1, +2, -1, -2} with probabilities 3/8, 1/4, 1/16, 1/4, 1/16 respectively) as 2ϵ.

We can continue this argument to find the differentials before step 1 and after step 8. We see that (after step 8) bits 1-0 and 31-16 of b_9 and d_9 have zero differential, those bits in c_9 has an ϵ differential, and those bits in a_9 has a 2ϵ differential. Now, both halves of the differential have the same block index (again, by the definition of a P/Q collision), and so the S table entries used at step 9 have zero differential. Since bits 1-0 and 31-17 of C has zero differential with probability 3/4, and A has zero differential in those same bits with probability 5/8, the corresponding bits output in step 9 will have zero differential with that same probability. In addition, the differential in B and D is isolated to bits 15-2, and so after step 9, the corresponding outputs in step 9 will have an additive differential that consists of either a number that is zero everywhere except in bits 15-2, or the negative of such a number[6]. We will shorthand such a differential as a differential of δ[7].

Summarizing the above, and listing the differentials found before step 1, we find the differentials which are listed in Table 2. Note that, in addition to the listed differentials, a_{output} has a zero differential in bits 1-0,31-16 with probability 3/8, and that c_{output} has a zero differential in bits 1-0,31-16 with probability 1/2. However, it turns out that using the differentials listed makes the attack somewhat more efficient.

Since P and Q are effectively 9 bit variables, and are updated 8 times with independent values, a P/Q collision has a 2^{-72} probability of occurring for two particular iterations with the same block index.

4 Searching for P/Q Collisions

To prosecute the attack, we must recognize with nontrivial probability when a P/Q collision has occurred. To do that, we search the keystream for places that resemble the differentials that occur immediately after a P/Q collision.

[4] Throughout this analysis, we will assume that the internal carries that occur within various additions will be unbiased and independently distributed.

[5] Because the carry from a sum of two random bits is biased towards zero, the additive differential will be somewhat more based towards 0 than stated. This effect turns out to make our attack more effective than expected, and so it can be safely ignored for our purposes.

[6] This number is biased towards zero, but this attack does not currently attempt to take advantage of that.

[7] Unlike other differentials discussed here, we consider a δ differential to be across the entire 32 bit variable.

Table 2. Known differentials before step 1 and at output at step 9 during a P/Q collision

Variable	Bits	Differential
a_1	19-2	Zero
b_1	10-2	ϵ
c_1	10-2	ϵ
d_1	10-2	ϵ
a_{output}	1-0,31-17	Zero with probability 5/8
b_{output}		δ
c_{output}	1-0,31-17	Zero with probability 3/4
d_{output}		δ

In a P/Q collision, Table 2 shows us that there are 34 output bits in a_{output} and c_{output} that agree with probability $5/8 \times 3/4 = 15/32$. In addition, the additive differential in b_{output} and d_{output} must both be one of $2^{15} - 1$ values. We will term any pair of outputs that meet the output criteria in Table 2, and which occur for the same nonfirst block index as a *potential P/Q collision.*

Assume that we have 2^{43} bytes of SEAL output that consist of the first two block indices from the entire SEAL output stream (that is, block indices 1 and 2 for all possible input strings and submessage indices). Then, we search through the outputs with block index 2 for pairs of output blocks that have the above properties. There are 2^{38} available output blocks with block index 2, and by the birthday paradox, and assuming that the internal states of the Ps and Qs are effectively random, we will have an expected $2^{2\times 38-72-1} = 8$ P/Q collisions, of which an expected $8 \times 15/32 \approx 4$ of which will be detectable in this manner.

We will assume that, if the internal state does not correspond to a P/Q collision, those output bits will match approximately as often as they will in a random sequence[8]. Then, we expect that an arbitrary pair will match if 17 bits within A and within C match, and the differences between B and D will be $2^{15}-1$ possible values (out of a total of 2^{32} possible values). Hence, the probability for an arbitrary pair of matching will be $2^{-17} \times 2^{-17} \times (2^{15} - 1)/2^{32} \times (2^{15} - 1)/2^{32} \approx 2^{-68}$. Hence, we expect approximately $2^{2\times 38-1-68} = 128$ false hits, that is, potential P/Q collisions that do not correspond to actual P/Q collisions.

The above analysis assumed that we have the first 32 bytes from every possible 1024 byte submessage. If we assume instead that we have a contiguous section of SEAL output (for example, the entire 64kbyte expansion of a subset of the input strings), then this attack can still be prosecuted, but with less efficiency, requiring 2^{45} bytes of output. The problem is that P/Q collisions are only possible within the same block, and hence increasing the number of block indices decreases the efficiency of the birthday paradox. There are 2^6 possible block indices (one of which is useless for searching for P/Q collisions), and with

[8] If this assumption is not valid, then potential P/Q collisions will be either much more numerous or much less numerous than expected, which can itself be used as a distinguisher.

2^{41} output blocks, we have 2^{35} output blocks for each index. Hence, each block index is expected to have $2^{2\times35-72-1} = 1/8$ P/Q collisions, of which an expected $1/8 \times 15/32 \approx 1/16$ will be detectable. In addition, each block index is expected to have $2^{2\times35-68-1} = 2$ false hits. Hence, we expect $1/16 \times 63 \approx 4$ true hits throughout the stream, and an expected $2 \times 63 = 126$ false hits. We can then continue to prosecute the attack as shown in section 5.

5 Verifying P/Q Collisions

Now that we have identified potential places where a P/Q collision may have occurred, we then examine the differential in the output states that occur at the corresponding block 1 to see if they resemble the output state immediately before a P/Q collision. If we look at the state of the cipher immediately before a P/Q collision, we see in Table 2 that bits 10-2 of a_1 have zero differential[9], and bits 10-2 of b_1, c_1, d_1 has an ϵ differential. Stepping back through steps 11 and 9 of the previous iteration, and canceling out the effects of the S array[10] by exclusive-oring the A and C outputs and subtracting the B and D outputs, the attacker observes[11]:

$$O_A = A_{Output} \oplus \hat{A_{Output}} = (A_1 - n_1) \oplus (\hat{A_1} - \hat{n_1}) \tag{1}$$

$$O_C = C_{Output} \oplus \hat{C_{Output}} = (C_1 \oplus n_1) \oplus (\hat{C_1} \oplus \hat{n_1}) \tag{2}$$

$$O_B = B_{Output} - \hat{B_{Output}} = (B_1 - n_2) - (\hat{B_1} - \hat{n_2}) \tag{3}$$

$$O_D = D_{Output} - \hat{D_{Output}} = (D_1 \oplus n_2) - (\hat{D_1} \oplus \hat{n_2}) \tag{4}$$

where O_A, O_B, O_C, O_D are defined as above, X_1, $\hat{X_1}$ are the contents of the X variable in the two sides of the differential immediately after step 11, n_1, n_2, $\hat{n_1}$, $\hat{n_2}$ are the values used in step 11 on the two sides, and X_{Output}, $\hat{X_{Output}}$ are the values corresponding to the variable X output by the two sides at step 9.

By assuming that the various unknown quantities above are independently and uniformly distributed, we can compute the expected distribution of O_A, O_B, O_C, O_D. When we do so, we find that the distribution is strongly skewed from uniform. For example, the output is within a subset that is only 1.113% of the size of the entire distribution 50% of the time. The most probable pattern (bits 10-2 of O_A, O_B, O_C, O_D are all zero) occurs with a probability 25,000 times the expected rate. In addition, nearly 37% of the possible O_A, O_B, O_C, O_D values are impossible for any value of unknown quantities.

[9] As do bits 19-11, but those bits cannot be used in our attack.

[10] We note again that both sides of the differential use the same block index, and so the S array contents here again have zero differential.

[11] Depending on whether the block index is even or odd, equations (1)-(4) may actually depend on n_3 and n_4 rather than n_1 and n_2. This attack does not depend on the actual identity of the variables, and so by convention we will call those variables n_1 and n_2.

This happens because exclusive-or and addition/subtraction are similar operations, and produce related results on a bit level more often than random operations would. For example, if you examine O_A and O_C, and ignore the effects of ϵ_1 and ϵ_3 (which, with good probability, affects only the lower order bits), then the equations (1) and (2) reduce to (over bits 10-0):

$$O_A \approx (X - n_1) \oplus (X - \hat{n_1}) \tag{5}$$

$$O_C \approx n_1 \oplus \hat{n_1} \tag{6}$$

where the approximation means that each bit has a probability of matching the corresponding bit of the other side, and the agreement probability is best at bit 10 and lesser as you go down. Now, consider the case where several consecutive bits of O_C are zero. This implies that the corresponding bits of n_1 and $\hat{n_1}$ agree with good probability. This, in turn, implies that the higher order bits of the corresponding section of $X - n_1$ and $X - \hat{n_1}$ will also agree with good probability, and hence a sequence of consecutive bits of O_C being zero tend to imply that the higher order bits of the corresponding sequence of O_A will also be zero.

There are similar, but weaker, relationships between O_B and O_D that can also be exploited. Rather than explicitly listing all these relationships, we can take advantage of this by computing the probability distribution that is formed by a source that generates the distribution expected by (O_A, O_B, O_C, O_D) with probability 4/128, and an equidistributed pattern with probability 124/128 (which we have shown is what is observed with SEAL), and compare that with the probability distribution of a source that generates an equidistributed pattern all the time (which is what a truly random source produces). If we use a Neyman-Pearson test ([1]), we are able to distinguish the two sources with 90% certainty with approximately 110 outputs. And, since the examined SEAL output is expected to have at least that many detected outputs, we can use that distinguisher to distinguish SEAL from a random source.

In an actual implementation of this attack, the bulk of the work (searching for potential P/Q collisions) can be done by sorting the adjacent output blocks on the bits that are known to match (bits 1-0,31-17 of a_{output}, c_{output} of the second output block, and block index), and then b_{output} value of the second output block. This sort will bring the potential P/Q collisions near each other, and so one sequential scan can find them. Once the potential P/Q collisions have been found, the computations required to verify the collisions are comparatively trivial. Hence, the run time is dominated by the time to do the sort, which is $O(2^{38} \log 2^{38}) \approx O(2^{43})$.

6 Experimental Verification on Weaked SEAL Variants

To be able to verify experimentally with the available computer resources that this attack is valid, we weakened SEAL by creating a variant, which we will refer to as wSEAL/5. This is defined by modifying each T table lookup and circular

rotates in the next state function to use only 5 bits, rather than 9. This change was done specifically to allow the attack to be run on the available hardware, while attempting to retain the essential aspects of the cipher.

The distinguisher translates in a straightforward manner to wSEAL/5. The only nonobvious change is that, when we recomputed the (O_A, O_B, O_C, O_D) distributions for these variants, they were considerably more uniform than the distribution for SEAL. This occurs because the bias in the distribution was due to relationships between the higher order bits, and when we reduce the size of the T table lookups, we also reduce the number of higher order bits that are available for verification. Because of the increased uniformity, it turns out that we require 800 potential P/Q collisions to achieve the same confidence level for wSEAL/5.

When we analyze these variants to determine if the assumptions that we made during our analysis is valid, we find:

- P/Q collisions are produced at the predicted rate. For example, when searching through 64 separate sections of 2^{20} blocks of wSEAL/5 output, we found 30 P/Q collisions, when the analysis would predict that 32 collisions were expected.
- P/Q collisions are proceeded by the a_1, b_1, c_1, d_1 differentials listed in Table 2. This was verified by examining the differentials from the 30 P/Q collisions referenced above.
- P/Q collisions that are detectable using the criteria of section 4 were produced at least the expected rate. For example, when searching through the 30 P/Q collisions referenced above, we found that 21 of them would be detected by that criteria. The analysis predicted that 15 were expected.
- Potential P/Q appear in the output stream at the expected rate. For example, in the above output stream that produced 21 detectable P/Q collisions, a total of 529 potential P/Q collisions were detecteds. The analysis predicted that 527 were expected[12].
- The P/Q collisions within the output stream are verifiable using the criteria of section 5. In particular, wSEAL/5 could be reliably distinguished from RC4 output using this analysis. In 100 tests where the distinguisher was presented either wSEAL/5 or an unrelated function, it correctly identified the source 88 times, which is consistent with our 90% confidence criteria we used.

Because wSEAL/5 acts in a manner consistent with our analysis, it appears reasonable to suppose that this analysis also applies to the full SEAL cipher.

7 Conclusions

We have presented a method for distinguishing SEAL from a random keystream source with approximately 2^{43} output bytes (or alternatively, with 2^{45} consecutive output bytes). Our method is the only such method in the open literature that is able to do so.

[12] This is 512 false hits, and 15 actual detectable P/Q collisions.

The attack appears to work by taking advantage of several aspects of the SEAL next state function:

- The actual number of sbox inputs are relatively small compared to the number of bytes actually output. This fact is listed in [6] as one of the reasons for its efficiency, however, we were able to use this property to control inputs to the sboxes.
- In line 11 of the SEAL update function, two of the n_i variables are used twice. The attack was able to use this redundancy to be able to verify when a collision occurred, by essentially verifying that the same values were used to update the internal variables. Had SEAL used all four n_i variables to update the four internal variables, there would have been no way to verify a P/Q collision.
- Lines 1 through 8 has very good differential propagation in the forward direction, but comparatively poor propagation in the reverse direction. In particular, if you track the effect of a difference, the output of an sbox in one step is immediately fed into the sbox on the next step. However, if you assume a difference at one step, and trace the differential backwards by logically inverting the next state function, you often skip several steps before the effect of an sbox lookup is used. We suspect that this is one of the reasons why this differential happens to work, and that a different structure that had reasonably good differential properties in both directions would be rather more resistant to similar attacks.

It will require more research to determine whether this attack can be modified to require less keystream, and whether it can be extended to rederive the secret key or to gain information on entries in the R, S, T tables.

References

1. Blahut, R., "Principles and Practice of Information Theory", Addison-Wesley, 1983.
2. O. Goldreich, S. Goldwasser, S. Micali, "How to construct random functions", Journal of the ACM, Vol 33, No. 4, 1986, pp. 210-217
3. H. Handschuh, H. Gilbert, "χ^2 cryptanalysis of the SEAL encryption algorithm", Fast Software Encryption, Lecture Notes in Computer Science, Vol. 1267, Springer-Verlag, 1997, pp. 1-12
4. A. Menezes, P. van Oorschot, S. Vanstone, "Handbook of applied cryptography", CRC Press, 1997
5. P. Rogaway, D. Coppersmith, "A software-optimized encryption algorithm", Fast Software Encryption, Lecture Notes in Computer Science, Vol 809, Springer-Verlag, pp. 56-63.
6. P. Rogaway, D. Coppersmith, "A software-optimized encryption algorithm", Journal of Cryptography, Vol. 11, No. 4, 1998, pp. 273-287

Cryptanalysis of SBLH

Goce Jakimovski and Ljupčo Kocarev

Institute for Nonlinear Science
University of California, San Diego
9500 Gilman Drive,
La Jolla, CA 92093-0402
lkocarev@ucsd.edu

Abstract. SBLH is a 256-bit key stream cipher that is used in Business Security's products for voice, fax and data communication. The cipher is claimed to be quite unique and yet very powerful. In this brief report, we suggest a possible chosen ciphertext attack on SBLH. We show that with 2^{24} ciphertext/plaintext pairs, one can successfully recover the active key of length 2^{17} bits.

1 Introduction

SBLH is a patented stream encryption algorithm developed by Business Security [1]. One may summarize the description of the algorithm in [1] using the following quote:

> *The algorithm has been in use in our products nearly ten years, during which it has been extensively analyzed by ourselves, our customers and various governments. All analytical effort pointed to the same conclusion, SBLH is a strong algorithm.*

The structure of the algorithm is very simple. It consists of four building blocks: two memories and two encoders. In early versions of systems built around SBLH, the memories were filled with random data generated by hardware source. In modern versions, a 256-bit key is used by the key expansion algorithm to create the active key. In this paper we show that with 2^{24} ciphertext/plaintext pairs, one can successfully recover the active key in SBLH of length 2^{17} bits. This is the outline of the paper. In the next section we present a general description of the algorithm, while in Section 3 we show how the active key of SBLH can be recovered using a chosen ciphertext attack.

2 General Description of SBLH

SBLH is a stream cipher i.e. the algorithm converts plaintext to ciphertext one bit at a time. It generates a stream of pseudorandom bits, which is XOR-ed with the plaintext to form the ciphertext. The ciphertext is then fed back into the algorithm making the output dependent on both the key and the previous

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 144–151, 2002.

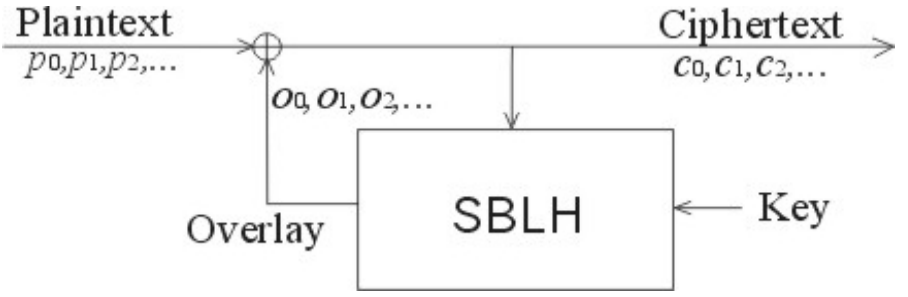


Fig. 1. The encryption process in SBLH

ciphertext. Figure 1 gives an outline of how the encryption process works in SBLH. Overlay is just another word for keystream.

As previously mentioned, each keystream bit is a function of the key and some of the previous ciphertext bits. More specifically each keystream bit is a function of the key and the 64 previous ciphertext bits. Thus, the algorithm is self-synchronizing. If a bit error or a bit slip occurs this will only affect the following 64 bits. After that, the algorithm will be synchronized again. The price we have to pay for self-synchronization is error expansion. Any single error will always affect the following 64 bits. Decryption with SBLH is very similar to encryption. The same keystream used during the encryption is once more added to the ciphertext. This produces the original plaintext. In order to reproduce the keystream used to encrypt, we put the XOR after the data is fed into SBLH. The data that was fed back into SBLH during the encryption is now fed into SBLH during the decryption. The algorithm will therefore produce the same keystream. The decryption process is shown in Figure 2.

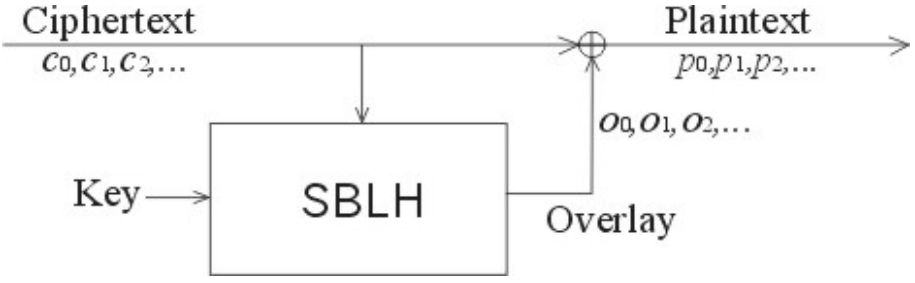


Fig. 2. The decryption process in SBLH

The internal structure of SBLH is quite unique, and the concept is very simple. SBLH is constructed from four simple building blocks, two memories and two encoders. This concept can be used in a wide variety of different algorithms. Key size, error propagation and self-synchronization are some of the parameters

that can be easily adjusted by modifying the basic building blocks of SBLH. Figure 3 describes the structure of SBLH working in encryption mode.

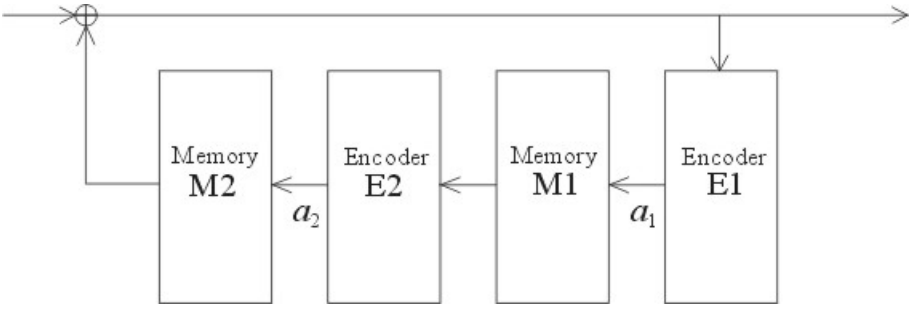


Fig. 3. The structure of SBLH

SBLH takes as input a bit stream (in figure 3 the bit stream is the ciphertext). A bit from the input stream enters the first encoder E_1 where it remains for some time T_1. During this time the bit and $n-1$ other bits currently in E_1 are used by the encoder to produce a sequence of addresses in memory M_1. A equivalent way of modeling E_1 is as a function of n bits, $E_1(x_0...x_{n-1}) = a_1$, where a_1 is an address in memory M_1.

The value stored in M_1 at the address a_1, generated by E_1, is fetched and passed to the second encoder E_2. E_2 is very similar to E_1. The second encoder E_2 is a function of m bits, $E_2(x_0...x_{m-1}) = a_2$, where a_2 is an address in memory M_2. The result of E_2 is used to fetch a value from M_2. The value at address a_2 of memory M_2 is used as overlay to produce the ciphertext. The error propagation and self-synchronization properties of SBLH are completely determined by the depth of the convolutional encoders E_1 and E_2. In this case, the encoder E_1 has 16 bits memory; the output from E_1 is dependent on the last 16 bits. The second encoder E_2 has 48 bits memory. Together E_1 and E_2 have 64 bits of memory, which equals the error propagation/self synchronization of SBLH. The conclusion is that the size of the memory of SBLH matches the sum of the sizes of memories in the encoders E_1 and E_2.

The memories M_1 and M_2 must each be filled with 2^{16} bits of random data. The contents of M_1 and M_2 form the active key. In early versions of systems built around SBLH, the memories were filled with random data generated by hardware source. This made the key length equal to 2^{16} bits. In modern versions the memories are filled with pseudorandom data. The actual key length has been reduced to 256 bits, which is expanded into memories M_1 and M_2. An overview of the key expansion can be found in figure 4.

The expansion starts with 256-bit random key being loaded into a memory. Then the shift-register is loaded with initial value. Before the algorithm can produce any output it has to perform a run-up (the algorithm goes through a sequence of steps without producing any output).

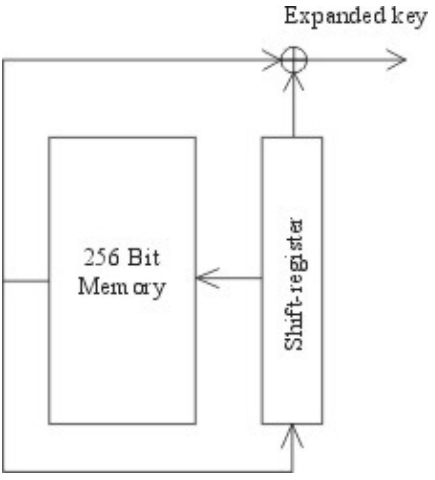


Fig. 4. The key expansion algorithm

3 Chosen Ciphertext Attack

In this section we will describe a chosen ciphertext attack on SBLH. We use the following notation:

c_i – the ciphertext bit in the i-th moment;

$a_{1,i} = E_1(c_{i-15} \dots c_i)$ – output of the encoder E_1 in the i-th moment;

$s_i = M_1(a_{1,i})$ – bit stored at address $a_{1,i}$ in the memory M_1;

$a_{2,i} = E_2(s_{i-47} \dots s_i)$ – output of the encoder E_2 in the i-th moment;

$o_i = M_2(a_{2,i})$ – overlay bit i.e. bit stored at address $a_{2,i}$ in the memory M_2.

Now, let us consider two ciphertext sequences $c'_0, c'_1, \dots c'_n$ and $c''_0, c''_1, \dots c''_n$ $(n \geq 63)$, such that

$$s'_i = s''_i,\ i = n-47, \dots, n-1. \tag{1}$$

If $M_1(a'_{1,n}) = M_1(a''_{1,n})$ i.e. the bit stored at address $a'_{1,n} = E_1(c'_{n-15} \dots c'_n)$ is equal to the bit stored at address $a''_{1,n} = E_1(c''_{n-15} \dots c''_n)$, then $s'_n = s''_n$. Therefore, $a'_{2,n} = E_2(s'_{n-47} \dots s'_n)$ equals $a''_{2,n} = E_2(s''_{n-47} \dots s''_n)$ and $o'_n = M_2(a'_{2,n}) = M_2(a''_{2,n}) = o''_n$. If $M_1(a'_{1,n}) \neq M_1(a''_{1,n})$ i.e. the bit stored at address $a'_{1,n} = E_1(c'_{n-15} \dots c'_n)$ and the bit stored at address $a''_{1,n} = E_1(c''_{n-15} \dots c''_n)$ are different, then the equality $o'_n = o''_n$ will hold with some probability δ. We will assume that δ has some fixed value less than one. If the number of zeroes in the memory M_2 equals the number of ones and $a_{2,i}$ is uniformly distributed, then $\delta = \frac{1}{2}$.

Example 1. The most obvious way to achieve $s'_i = s''_i, i = n-47, \dots, n-1$ is to choose the ciphertext sequences such that $c'_i = c''_i, i = 0, \dots, n-1$. Let $c'_i = c''_i = 1, i = 0, \dots, n-1$, $c'_n = 1$ and $c''_n = 0$. In this case, the condition (1) is satisfied, $a'_{1,n} = E_1(1111111111111111)$, $a''_{1,n} = E_1(1111111111111110)$, $s'_n = M_1(a'_{1,n})$ and $s''_n = M_1(a''_{1,n})$. If $M_1(a'_{1,n}) = M_1(a''_{1,n})$, then $s''_n = s'_n \Rightarrow a''_{2,n} = a'_{2,n} \Rightarrow$

$o''_n = o'_n$. If $M_1(a'_{1,n}) \neq M_1(a''_{1,n})$, then $o''_n = o'_n$ iff $M_2(E_2(s''_{n-47} \dots s''_n)) = M_2(E_2(s'_{n-47} \dots s'_n))$. The last equation is satisfied with some probability δ.

The previously described property of the algorithm can be exploited to check whether the bits at some addresses $A_1 = a'_{1,n}$ and $A_2 = a''_{1,n}$ in the memory M_1 are equal or not. We submit for decryption two ciphertext sequences c'_i and $c''_i, i = 0, \dots 63$, such that $s'_i = s''_i, i = 16, \dots 62$, $E_1(c'_{48} \dots c'_{63}) = A_1$ and $E_1(c''_{48} \dots c''_{63}) = A_2$ (we'll not discuss the existence of the sequences, because for the addresses that will be used in the attack it is easy to construct such sequences). From the ciphertext and corresponding plaintext sequences we compute o'_{63} and o''_{63} . If $o'_{63} \neq o''_{63}$, then $s'_{63} \neq s''_{63}$ i.e. the bit at address A_1 and the bit at the address A_2 are different. If $o'_{63} = o''_{63}$, then we are not sure whether $M_1(A_1) = M_1(A_2)$ or $M_1(A_1) \neq M_1(A_2)$ (as mentioned before, we'll assume that the probability of the event $o'_{63} = o''_{63}$, when $M_1(A_1) \neq M_1(A_2)$ is δ). Hence, if $o'_{63} = o''_{63}$ we repeat the previously described procedure. We repeat the procedure until event $o'_{63} \neq o''_{63}$ happens. If, after large number of repetitions, $o'_{63} = o''_{63}$ is always satisfied, then we assume that $M_1(A_1) = M_1(A_2)$ i.e. the bit at address A_1 in the memory M_1 is equal to the bit at address A_2 in the memory M_1. The probability that we made a mistake is $\delta^t \to 0, t \to \infty$, where t is the number of repetitions of the procedure. We will assume that $M_1(A_1) = M_1(A_2)$ if $o'_{63} = o''_{63}$ during $t = 256$ repetitions.

Example 2. Suppose we want to check whether the bits at addresses A_1 and A_2, where $A_1 = E_1(1111111111111111)$ and $A_2 = E_2(1111111111111110)$, are equal. We submit for decryption the following ciphertext sequences:

$$\underbrace{1111\dots11}_{40bits}1010101011111111111111$$

$$\underbrace{1111\dots11}_{40bits}1010101011111111111110$$

From the resultant plaintexts we compute o'_{63} and o''_{63}. If $o'_{63} \neq o''_{63}$, then we assume that $M_1(A_1) \neq M_1(A_2)$. If $o'_{63} = o''_{63}$, then we submit for decryption two new ciphertext sequences:

$$\underbrace{1111\dots11}_{40bits}1110101011111111111111$$

$$\underbrace{1111\dots11}_{40bits}1110101011111111111110$$

From the resultant plaintexts we compute the new overlay bits o'_{63} and o''_{63}. If $o'_{63} \neq o''_{63}$, then we assume that $M_1(A_1) \neq M_1(A_2)$, while if $o'_{63} = o''_{63}$, we repeat the procedure. If, after 256 repetitions, $o'_{63} = o''_{63}$ is always satisfied, we assume that $M_1(A_1) = M_1(A_2)$.

Let $M_1(A_1) = M_1(A_2)$. We can use this fact to determine the relation between bits at some addresses A'_1 and A'_2. This is done in the next example.

Example 3. Let us assume that during the procedure described in example 2 it was determined that $M_1(A_1) = M_1(A_2)$. We can also determine the relation between $M_1(A'_1)$ and $M_1(A'_2)$, where $A'_1 = E_1(1111111111111110)$ and $A'_2 = E_1(1111111111111100)$. Since $s'_{63} = s''_{63}$ for all ciphertext sequences used in the previous example, we can construct the following procedure for checking whether the bits at addresses A'_1 and A'_2 are equal or not. Submit for decryption two ciphertext sequences:

$$\underbrace{1111\ldots11}_{40bits}10101010111111111111111110$$

$$\underbrace{1111\ldots11}_{40bits}10101010111111111111111100$$

From the resultant plaintexts compute o'_{64} and o''_{64}. If $o'_{64} \neq o''_{64}$, then $M_1(A'_1) \neq M_1(A'_2)$. If $o'_{64} = o''_{64}$, then submit for decryption two new ciphertext sequences:

$$\underbrace{1111\ldots11}_{40bits}11101010111111111111111110$$

$$\underbrace{1111\ldots11}_{40bits}11101010111111111111111100$$

From the resultant plaintexts we compute the new overlay bits o'_{64} and o''_{64}. If $o'_{64} \neq o''_{64}$, then we assume that $M_1(A'_1) \neq M_1(A'_2)$, while if $o'_{64} = o''_{64}$, we repeat the procedure. If, after 256 repetitions, $o'_{64} = o''_{64}$ is always satisfied, we assume that $M_1(A'_1) = M_1(A'_2)$.

In the examples 2 and 3, it was shown how relations between bits of the memory M_1 could be determined. Each new relation that can't be derived from the previous relations reveals one bit of the contents of M_1. More relations we construct, more information about the contents of M_1 we gain. If the number of relations is large enough, we can determine all 2^{16} bits of M_1. In the following, we describe systematic procedure for constructing relations between the bits of M_1.

0. Initialize an array r of $2^{16} \times 2^{16}$ elements. The elements of r contain information about the relations between the bits of M_1. If the relation between the bit at address $E_1(a)$ and the bit at address $E_1(b)$ is not determined, then $r[a][b] = r[b][a] = 2$. If the relation between the bit at address $E_1(a)$ and the bit at address $E_1(b)$ is determined and $M_1(E_1(a)) = M_1(E_1(b))$, then $r[a][b] = r[b][a] = 1$. If the relation between the bit at address $E_1(a)$ and the bit at address $E_1(b)$ is determined and $M_1(E_1(a)) \neq M_1(E_1(b))$, then $r[a][b] = r[b][a] = 0$. In this step, it holds $r[a][b] = 2(a \neq b)$ and $r[a][b] = 1(a = b)$.

1. According to example 2, for all a and b such that $a \oplus b = 0000000000000001$ find out how the bits at addresses $E_1(a)$ and $E_1(b)$ are related. If $M_1(E_1(a)) = M_1(E_1(b))$, then set $r[a][b] = r[b][a] = 1$. If $M_1(E_1(a)) \neq M_1(E_1(b))$, then set $r[a][b] = r[b][a] = 0$.

2. According to example 3, for all a and b such that $r[a][b] = 1$ and $a \oplus b = 0000000000000001$ we can determine the relation between $M_1(E_1(a'))$ and

$M_1(E_1(b'))$, where $a' = (a << 1) \bmod 2^{16}$ and $b' = (b << 1) \bmod 2^{16}$. We note that $a' \oplus b' = 0000000000000010$. If the relation between $M_1(E_1(a'))$ and $M_1(E_1(b'))$ is already determined ($r[a'][b'] \neq 2$), do nothing. If $r[a'][b'] = 2$, then find out how the bits at addresses $E_1(a')$ and $E_1(b')$ are related and update the array r. By updating the array r we mean entering the new relation and all the relations that can be derived using the new and the previous relations.

3. The fourth step is analogous to the previous step. In this step, for all a and b such that $r[a][b] = 1$ and $a \oplus b = 000000000000001\star$ ($\star$ denotes arbitrary value) we can determine the relation between $M_1(E_1(a'))$ and $M_1(E_1(b'))$, where $a' = (a << 1) \bmod 2^{16}$ and $b' = (b << 1) \bmod 2^{16}$. We note that $a' \oplus b' = 00000000000001\star 0$. If the relation between $M_1(E_1(a'))$ and $M_1(E_1(b'))$ is already determined ($r[a'][b'] \neq 2$), do nothing. If $r[a'][b'] = 2$, then find out how the bits at addresses $E_1(a')$ and $E_1(b')$ are related and update the array r.

$\vdots$

16. In a similar way, new relations between bits at addresses $E_1(a')$ and $E_1(b')$ are derived using the relations determined in the previous step. For a' and b' holds that $a' \oplus b' = 1\star\star\star\star\star\star\star\star\star\star\star\star\star\star 0$.

17. In this step, the entries of the encoder E_1 are grouped into classes using the array r. The grouping is performed so that the number of classes is minimal. Let N denotes the number of classes. Each class $K_i, i = 1, \ldots N$ consists of two subsets K_i^1 and K_i^0. The following must hold: (i) $a, b \in K_i^1$ (or $a, b \in K_i^0$) iff $r[a][b] = r[b][a] = 1$, (ii) $a \in K_i^1$ and $b \in K_i^0$ iff $r[a][b] = r[b][a] = 0$, (iii) $a \in K_i$ and $b \in K_j$, $i \neq j$ iff $r[a][b] = r[b][a] = 2$.

18. 2^N possible contents of M_1 are constructed using the classes K_i. For each of these contents, the content of M_2 is determined and tested using the existing ciphertext/plaintext pairs. If the content of M_2 can be uniquely determined i.e. there is no address A such that in one moment $M_2(A) = 0$ and in the other $M_2(A) = 1$, we consider the pair (M_1, M_2) as a possible value of the active key.

The efficiency of the attack depends on the number of classes N. Suppose that the entries of the encoder E_1 are grouped into classes after each step. Let N_i denotes the number of classes after i-th step, let $l_i = \frac{2^{16}}{N_i}$ denotes the average class length and let L_i denotes maximum possible average class length after the i-th step. We note that $L_1 = 2, L_2 = 4, \ldots, L_{16} = 2^{16}$. Figure 5 depicts the values of $D(i) = L_i - \bar{l_i}$ and $d(i) = \frac{D(i)}{L_i}$, where $\bar{l_i}$ is an arithmetic mean of l_i calculated for 2^{10} randomly chosen contents of M_1.

It is obvious that the difference D tends to zero. This is due to the fact that the average number of elements of the classes increases with every step, and thus, the number of relations that can be derived increases too. On the other hand, the required number of new relations decreases with every step. Therefore, the number of contents of M_1 for which $N > 1$ is negligibly small.

When $N = 1$, there are only two possible contents of M_1. Therefore, the active key is one of the two possible pairs (M_1, M_2). Which one, it can be easily checked. The number of relations needed to group the entries of the encoder

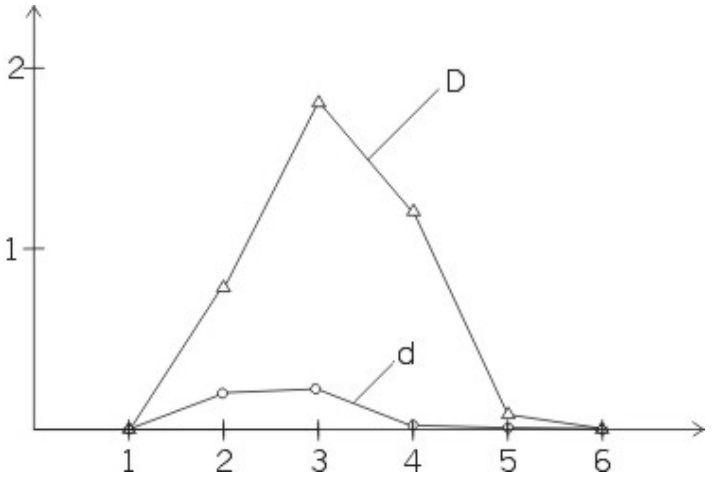


Fig. 5. $D(i)$ and $l(i)$ versus i. See the text for details.

E_1 in one class is $2^{16} - 1$. The number of required ciphertext/plaintext pairs is $(2^{16} - 1) \times 256 \approx 2^{24}$.

References

1. Business Security AB, Lund, Sweden, http://www.bsecurity.se

A Practical Attack on Broadcast RC4

Itsik Mantin and Adi Shamir

Computer Science Department, The Weizmann Institute, Rehovot 76100, Israel.
{itsik,shamir}@wisdom.weizmann.ac.il

Abstract. RC4 is the most widely deployed stream cipher in software applications. In this paper we describe a major statistical weakness in RC4, which makes it trivial to distinguish between short outputs of RC4 and random strings by analyzing their second bytes. This weakness can be used to mount a practical ciphertext-only attack on RC4 in some broadcast applications, in which the same plaintext is sent to multiple recipients under different keys.

1 Introduction

A large number of stream ciphers were proposed and implemented over the last twenty years. Most of these ciphers were based on various combinations of linear feedback shift registers, which were easy to implement in hardware, but relatively slow in software. In 1987 Ron Rivest designed the RC4 stream cipher, which was based on a different and more software friendly paradigm. It was integrated into Microsoft Windows, Lotus Notes, Apple AOCE, Oracle Secure SQL, and many other applications, and has thus become the most widely used software-based stream cipher. In addition, it was chosen to be part of the Cellular Digital Packet Data specification. Its design was kept a trade secret until 1994, when someone anonymously posted its source code to the Cypherpunks mailing list. The correctness of this unofficial description was confirmed by comparing its outputs to those produced by licensed implementations.

RC4 has a secret internal state which is a permutation of all the $N = 2^n$ possible n bits words. The initial state is derived from a variable size key by a key scheduling algorithm, and then RC4 alternately modifies the state (by exchanging two out of the N values) and produces an output (by picking one of the N values).

In practical applications n is typically chosen as 8, and thus RC4 has a huge state of $log_2(2^8!) \approx 1684$ bits. It is thus impractical to guess even a small part of this state, or to use standard time/memory/data tradeoff attacks. In addition, the state evolves in a complex non-linear way, and thus it is difficult to combine partial information about states which are far away in time. Consequently, all the techniques developed to attack stream ciphers based on linear feedback shift registers seem to be inapplicable to RC4.

Since RC4 is such a widely used stream cipher, it had attracted considerable attention in the research community, but so far no one had found an attack on

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 152–164, 2002.

RC4 which is even close to being practical: For $n = 8$ and sufficiently long keys, the best known attack requires more than 2^{700} time to find its initial state.

Our main result in this paper is the discovery of a trivial distinguisher between RC4 and random ciphers, which needs only two output words under several hundred unknown and unrelated keys to make a reliable decision (the best previously published distinguisher requires more than a billion output words). This is surprising, since the algorithm had been analyzed internally for more than 13 years and externally for more than 6 years, and this obviously nonrandom behavior should have been discovered long ago. Since RC4 has such a unique "fingerprint", it is easy to recognize its use in given black box encryption devices. In addition, the nonrandom behavior of RC4 makes it possible to deduce partial information about the plaintext by analyzing a small number of ciphertexts produced from the same plaintext under unrelated and arbitrarily long keys. This provides a practical ciphertext-only attack on RC4 in many applications which allow data broadcasting, such as groupware and email.

The paper is organized in the following way: In section 2 we describe RC4 and survey previous results about its security. In section 3 we show that some RC4 outputs are highly nonuniform, and analyze the number of outputs required to turn this nonuniformity into a reliable distinguisher. In addition, we show how to apply a practical ciphertext-only attack on RC4 in broadcast applications. In section 4 we generalize our particular observation, and show that such biases are created by *predictive* states, which are related to (but more general than) the fortuitous states introduced in [FM00]. We end the section with several examples of such states, along with a theoretical analysis of the size of biases they can generate and the complexity of the distinguishers they give rise to.

2 RC4 and Its Security

2.1 Description of RC4

The RC4 stream cipher is still considered a trade secret, and the following unofficial description is taken from Schneier's book [Sch96]. It consists of a key scheduling part (which turns a random key whose typical size is 40-256 bits into an initial permutation S of $\{0, \ldots, N-1\}$, and is ignored in this paper) and an output generation part which is described in figure 1. It initializes two indices i and j to 0, and then loops over four simple operations which increment i as a counter, increment j pseudo randomly, exchange the two values of S pointed to by i and j, and output the value of S pointed to by $S[i] + S[j]$[1]. Note that every entry of S is swapped at least once (possibly with itself) within any N consecutive rounds, and thus the permutation S evolves fairly rapidly during the output generation process.

[1] Here and in the rest of the paper all the additions are carried out modulu N

Initialization:
$i = 0$
$j = 0$
Generation loop:
$i = i + 1$
$j = j + S[i]$
Swap$(S[i], S[j])$
Output $z = S[S[i] + S[j]]$

Fig. 1. The Output Generation Algorithm

2.2 Previous Attacks on RC4

Interesting properties of RC4 were described in several papers. Finney specified in [Fin94] a class of states that RC4 can never enter. This class contains all the states for which $j = i + 1$ and $S[j] = 1$ (a fraction of approximately N^{-2} of the RC4 states are in this class, which is closed under RC4 round operation). These states are connected by short cycles of length $N(N - 1)$. Since the state transition in RC4 is invertible and the initial state ($i = j = 0$) is not of this type, RC4 can never enter these states for any key. Additional properties of the state transition graph of RC4 were analyzed by Mister and Tavares in [MT98].

Jenkins identified in [Jen] a probabilistic correlation between the secret information (S, j) and the public information (i, z) (in known message attacks), in which the events $S[j] = i - z$ and $j - S[i] = z$ occur with approximately twice their expected probability.

Andrew Roos noted in [Roo95] that for keys for which $K[0] + K[1] = 0$, the first output is equal to $K[2] + 3$ with probability $2^{-2.85}$. The cryptanalyst can use this fact to deduce two bytes of information about the key ($K[0] + K[1]$ and $K[2]$) with probability $2^{-10.85}$ instead of the trivial 2^{-16}, and thus reduce the effective key length in exhaustive search by about five bits.

Grosul and Wallach show in [GW00] that for large keys whose size is close to 2^n bytes, RC4 is vulnerable to a related key attack.

A branch and bound attack that is based on the "Guess on Demand" paradigm is analyzed in [MT98] and [KMP+98]. The attack simulates the generation process, and keeps track of all the known values in S which had been deduced so far. Whenever an unknown entry in S is needed in order to continue the simulation, the attacker tries all the possible values (their number is typically smaller than N since known values in the permutation S cannot be repeated). Actual outputs are used by the simulation to either deduce additional values in S (if the pointed value is unknown) or to backtrack (if the pointed value is known and different from the actual output). This tree search is simple to implement, and needs only $O(N)$ outputs. However, its enormous running time (which was analyzed both analytically and experimentally in [KMP+98]) makes

it worse than exhaustive search for typical key sizes, and completely impractical for any value of n above 5.

A different research direction was to analyze the statistical properties of RC4 outputs, and in particular to construct distinguishers between RC4 and truly random bit generators. Golić described in [Gol97] a linear statistical weakness of RC4, caused by a positive correlation between the second binary derivative of the lsb and 1. This weakness implies that RC4 outputs of length $2^{6n-7.8}$ ($2^{40.2}$ for the typical $n = 8$) can be reliably distinguished from random strings. This result was subsequently improved by Fluhrer and McGrew in [FM00]. They analyzed the distribution of triplets consisting of the two outputs produced at times t and $t + 1$ and the known value of $i \equiv t(modN)$, and found small biases in the distribution of $(7N - 8)$ of these N^3 triplets: some of these probabilities were $2^{-3n}(1 + 2^{-n})$ (with a small positive bias), and some of these probabilities were $2^{-3n}(1 - 2^{-n})$ (with a small negative bias). They used information theoretic methods to prove that these biases can be used to distinguish between $RC4$ and a truly random source by analyzing sequences of $2^{30.6}$ output words. They also identified a special class of states, which they denoted as fortuitous states, that are the source of most of these biases. These states can be used to extract part of the internal state with non-trivial probability, but since RC4 states are huge this does not lead to practical attacks on RC4 for $n > 5$.

3 The New Attack

Our main observation is that the second output word of RC4 has a very strong bias: It takes on the value 0 with twice the expected probability (1/128 instead of 1/256 for $n = 8$). Other values of the second output and all the values of other outputs have almost uniform distributions.

3.1 Distinguishing between Distinguishers

One interesting question is why hasn't this strong bias been discovered long ago? RC4 was subjected to rigorous statistical tests, but they typically consisted of taking a single stream with billions of output words, and counting the number of times each value occurred along it. Even if this experiment is repeated many times with different keys, the strong bias of the second output word is not going to be visible in such an experiment.

The difference between these types of statistical experiments is not new. Goldreich described in [Gol99] two different notions of computational indistinguishability, *indistinguishability in polynomial time* and *indistinguishability in polynomial sampling*. The first definition deals with a probabilistic polynomial time machine (called a *weak distinguisher*) which is given a single output stream produced by one of two possible sources, whereas the second definition deals with a probabilistic polynomial time machine (called a *strong distinguisher*) which is given as a black box one of the two sources, and can reset and rerun it with

new random keys polynomially many times. These notions of indistinguishability are theoretically equivalent in the sense that if one of them can succeed with probability $\frac{1}{2} + \frac{1}{p(\cdot)}$ for some polynomial p, then the other can also succeed with probability $\frac{1}{2} + \frac{1}{q(\cdot)}$ for some other polynomial q. However, in practice there can be a large difference between the two biases, and in particular one of them can be a large constant whereas the other is a tiny polynomial fraction. This phenomenon is wonderfully demonstrated in the case of RC4, since the analysis of a single output stream makes the bias almost unnoticeable, whereas the analysis of a single output word across many executions amplifies the bias and makes it almost unmissable.

3.2 The Biased Second Output of RC4

Theorem 1 *Assume that the initial permutation S is randomly chosen from the set of all the possible permutations of $\{0, \ldots, N-1\}$. Then the probability that the second output word of RC4 is 0 is approximately $2/N$.*

Proof: Denote the permutation S after it has been updated in round t by S_t (S_0 is the initial permutation) and the output of this round as z_t. We show that when $S_0[2] = 0$ and $S_0[1] \neq 2$, the second output is 0 with probability 1 (see figure 2).

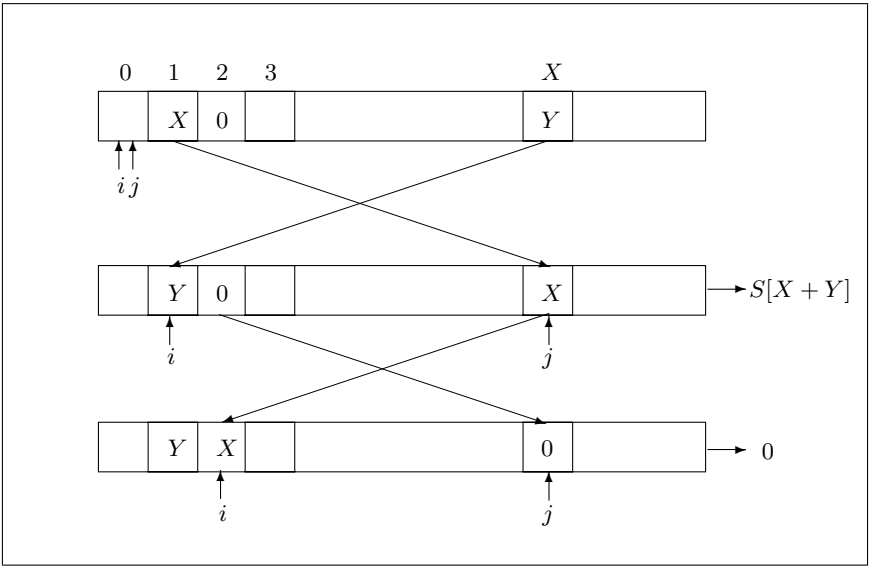


Fig. 2. The first 2 rounds of RC4 when $S_0[2] = 0$ and $S_0[1] \neq 2$

Initially, i and j are 0. If we denote $S_0[1]$ by X, then during the first round i is updated to 1 and j is updated to $0 + S_0[1] = X$, and the contents X and

Y of locations 1 and X are exchanged. The first output is $S_1[X+Y]$, which can be any value with essentially uniform probability distribution. We now use the assumption that $S_0[2]=0$. During the second round, i is incremented to 2, and j is incremented to $X+0=X$, and thus we exchange the values 0 and X of locations 2 and X, and output $S_2[X+0]=S_2[X]=0$ (we have to assume that $S_0[1]\neq 2$ since otherwise 0 is swapped out before it is used as an output). This lucky sequence of events proves that for about $1/N$ of the keys, the second output is 0 with probability 1, whereas for the other $1-1/N$ of the keys, the second output is 0 with probability $1/N$ (since it is uniformly distributed). As a result, the total probability that the second output is 0 is

$$P[z_2=0]=P[z_2=0|S_0[2]=0]\cdot P[S_0[2]=0]+P[z_2=0|S_0[2]\neq 0]\cdot P[S_0[2]\neq 0]$$

$$\approx 1\cdot 1/N+(1-1/N)\cdot 1/N=1/N\cdot(1+1-1/N)\approx 2/N$$

which is twice its expected probability. □

One could expect to see a similar (but weaker) bias towards 0 at all the other outputs z_t with $t=0 \mod n$), since in $1/N^2$ of these cases $S_t[2]=0$ and $j=0$, which would give rise to the same situation. However, extensive experiments have shown that this weaker bias at later rounds does not exist. By carefully analyzing this situation one can show that for any $j\neq 0$ the output is zero with a slight *negative* bias, and the total contribution of these negative biases exactly cancels the positive bias derived from $j=0$. The only time we don't have this cancellation effect is at the beginning of the execution, when j always starts as 0 rather than as a uniformly distributed random value.

An interesting observation based on this bias, is that by applying Bayes rule to the equation $P[z_2=0|S_0[2]=0]\approx 2/N$, we get

$$P[S_0[2]=0|z_2=0]=\frac{P[S_0[2]=0]}{P[z_2=0]}\cdot P[z_2=0|S_0[2]=0]\approx\frac{1/N}{2/N}\cdot 1=\frac{1}{2}$$

Consequently, whenever the second output byte is 0 we can extract an entry of S with probability $1/2$, which significantly exceeds the trivial probability of $1/N$. This fact can be used to accelerate most of the known attacks by a factor of $N/2$, but this improvement does not suffice to mount a practical attack on $\mathrm{RC4}_{n>5}$.

3.3 Cryptanalytic Applications

The strong bias described in section 3.2 has several practical cryptanalytic applications.

Distinguishing RC4 from Random Sources. The best distinguisher mentioned in the literature ([FM00]), distinguishes *RC4* from a random source by analyzing $2^{30.6}$ output words. This distinguisher is *weak* (since it analyzes a single output stream), and is based on $7N-8$ independent biases of events that

happens with probability of $2^{-3n}(1 \pm 2^{-n})$ instead of the expected 2^{-3n}. Our new observation can be used to construct a *strong* distinguisher for RC4 which requires only $O(N)$ output words.

Theorem 2 *Let X, Y be distributions, and suppose that the event e happens in X with probability p and in Y with probability $p(1+q)$. Then for small p and q, $O(\frac{1}{pq^2})$ samples suffice to distinguish X from Y with a constant probability of success.*

Proof: Let X_e, Y_e be the random variables specifying the number of occurrences of e in t samples. Then X_e and Y_e have binomial distributions with parameters (t, p) and $(t, p(1+q))$, and their expectations, variances and standard deviations are:

$$E[X_e] = tp, E[Y_e] = tp(1+q)$$

$$V(X_e) = tp(1-p) \approx tp, V(Y_e) = tp(1+q)(1-p(1+q)) \approx tp(1+q)$$

$$\sigma(X_e) = \sqrt{V(X_e)} \approx \sqrt{tp}, \sigma(Y_e) = \sqrt{V(Y_e)} \approx \sqrt{tp(1+q)} \approx \sqrt{tp}$$

We'll analyze the size of t that implies a difference of at least one standard deviation between the expectations of the two distributions:

$$E[Y_e] - E[X_e] \geq \sigma(X_e) \Leftrightarrow tp(1+q) - tp \geq \sqrt{tp} \Leftrightarrow tpq \geq \sqrt{tp} \Leftrightarrow t \geq \frac{1}{pq^2}$$

Consequently, $O(\frac{1}{pq^2})$ samples (The constant depends on the desired success probability) suffice for the distinguishing. □

Let X be the probability distribution of the second output in uniformly distributed streams, and let Y be the probability distribution of the second output in streams produced by RC4 for randomly chosen keys. The event e denotes an output value of 0, which happens with probability of $1/N$ in X and $2/N$ in Y. By using the previous theorem with $p = 1/N$ and $q = 1$, we can conclude that we need about $\frac{1}{pq^2} = N$ outputs to reliably distinguish the two distributions. To find the exact number, we carried out actual experiments, and found out that by analyzing just 200 streams we can correctly distinguish between RC4 outputs and random streams with probability of success which exceeds 0.64.

A Ciphertext-Only Attack on Broadcast RC4. A classical problem in distributed computing is to allow N Byzantine generals to coordinate their actions when up to one third of them can be traitors. The problem is solved by a multiround protocol in which each general broadcasts the same message (which initially consists of either "Attack" or "Retreat") to all the other generals, where each copy is encrypted under a different key agreed in advance between any two

generals. By using RC4, the generals will succeed in reaching a coordinated decision, but will probably fail in implementing it, since an enemy that collects all the ciphertexts can easily deduce which one of the two possible messages was broadcast by each general, regardless of the length of their keys. More formally, we show:

Theorem 3 *Let M be a plaintext, and let $C_1, C_2, \ldots, C_k$ be the RC4 encryptions of M under k uniformly distributed keys. Then if $k = \Omega(N)$, the second byte of M can be reliably extracted from $C_1, C_2, \ldots, C_k$.*

Proof: For every encryption key, $M[2]$ has probability $\frac{2}{N}$ to be XORed with 0, and probability $\frac{1}{N}$ to be XORed with each of the other possible bytes. Thus, a fraction of $\frac{2}{N}$ of the second ciphertext bytes are expected to have the same value as the second plaintext byte, and thus the most frequent character in $C_1[2], \ldots, C_k[2]$ is likely to be $M[2]$ itself.

□

The Byzantine empire no longer exists (did they use RC4?), but there are many other broadcasting protocols which are used today in a variety of applications. For example, many users send the same email message to multiple recipients (encrypted under different keys), and many groupware applications enable multiple users to synchronize their documents by broadcasting encrypted modification lists to all the other group members. All these applications are vulnerable to this attack.

4 Analysis

In this section we introduce the notion of *predictive* states, and show that it generalizes both our observation and the notion of fortuitous states which were defined in [FM00]. Each predictive state can give rise to some biased outputs, and we analyze the size of such biases as a function of certain parameters of these states. Finally, we show that such states can be used to improve the time and data complexities of branch and bound attacks on the internal state of RC4.

4.1 *a*-States and *b*-Predictiveness

Definition 1 *An* a-state *is a partially specified RC4 state, that includes i, j, and a (not necessarily consecutive) elements of S.*

Definition 2 *Let A be an a-state for some a. Suppose that all*[2] *the RC4 states that are compatible with A produce the same output word after r rounds. Then A is said to* predict *its rth output.*

[2] We can generalize the definition without affecting the analysis by allowing some exceptions due to rare coincidences.

Definition 3 *Let A be an a-state, and suppose that for some $r_1, \ldots, r_b \leq 2N$, A predicts the outputs of rounds $r_1, \ldots, r_b$. Then A is said to be* b-predictive[3]

Intuitively, a b-predictive a-state can be associated with a sequence of b specific (round, output) pairs. Many experiments (and strong intuition) indicate that an a-state can be b- predictive only when $a \geq b$, but formalizing a proof for this conjecture seems to be non-trivial.

4.2 Distinguishers Based on Predictive States

In general, any a-state that is b-predictive can induce some bias in the output distribution. Consider the events E_A (that the current state is compatible with the a-state A), and E_B (that the outputs in rounds $r_1, \ldots, r_b$ are those predicted by A). E_A includes $a+2$ constraints (i, j and a elements of S) and thus has a probability of $\frac{1}{N^2}\frac{N!}{(N-a)!} \approx N^{-(a+2)}$. Whenever E_A occurs, E_B occurs with probability 1, and whenever E_A does not occur, E_B typically happens with the trivial probability of $N^{-(b+1)}$ (based on the choice of i and the b outputs). Thus the probability of E_B is computed as follows

$$P[E_B] = P[E_B|E_A] \cdot P[E_A] + P[E_B|\neg E_A] \cdot P[\neg E_A] \approx$$

$$\approx 1 \cdot N^{-(a+2)} + N^{-(b+1)} \cdot (1 - N^{-(a+2)}) =$$

$$= N^{-(b+1)}(1 - N^{-(a+2)} + N^{b+1-(a+2)}) \approx N^{-(b+1)}(1 + N^{b-a-1})$$

Applying the result of Theorem 2 to these probabilities yields the following corollary

Corollary 1 *An a-state that is b-predictive implies the existence of a distinguisher for RC4 which requires $O(N^{2a-b+3})$ output words.*

Proof: In terms of Theorem 2, $p = N^{-(b+1)}$ and $q = N^{b-a-1}$. Thus, the number of output words required to reliably distinguish RC4 from random sources is $O(\frac{1}{pq^2}) = O(N^{2a-b+3})$ □

Our major observation in this paper was caused by a 1-predictive 1-state, $i_A = 0, j_A = 0, S_A[2] = 0$ which predicted an output of zero in the second round. Since at the beginning of the generation process i (which is part of the definition of p) and j (which is part of the definition of q) are fixed, we actually get a better distinguisher than the one implied by the general corollary: both p and q (from Theorem 2) increase by a factor of N, and thus the distinguisher needs only $O(N)$ output words.

[3] without the bound on the r_i's, this definition might include degenerate cases since the output of RC4 eventually cycles.

4.3 An Attack Based on Predictive States

Assuming the uniformity of RC4 internal state and outputs (their slight biases are negligible in the current context), a b-predictive a-state can be used to extract some partial information on the internal state with non-trivial probability. By applying Bayes Rule to the probabilities of the events E_A and E_B from section 4.2) we can calculate

$$P[E_A|E_B] = \frac{P[E_A]}{P[E_B]} P[E_B|E_A] \approx \frac{N^{-(a+2)}}{N^{-(b+1)}} \cdot 1 = N^{b-a-1}$$

Since E_A is internal and E_B is external, we can use the external event E_B as an indicator for the internal event E_A with a success probability of N^{b-a-1}. In other words, if we see N^{a+1-b} occurrences of E_B, one of them is likely to be a real occurrence of E_A. Having j and a values of S, we can apply the branch and bound attack from [KMP+98] to reveal the rest of the secret state of RC4.

Intuitively, a determines the quantity of the revealed information, but increasing a reduces the probability for E_A and increases the required number of outputs. Consequently, the attack is limited to small values of a. The value of $b - a$ determines the number of false candidates that have to be examined, and reducing it is essential to reducing the time complexity. Under the conjecture that $b \leq a$, the best states for this attack are a-predictive a-states. In order to reduce the required number of outputs, the attack can use a database of many a-predictive a-states sorted by outputs, and lookup all the output sequences in it.

The attack requires an efficient way to find in a preprocessing stage all the a-predictive a-states of RC4 for small values of a. This is a non-trivial problem whose complexity is not well understood at the moment. A subset of predictive states that is easier to enumerate is the class of fortuitous states mentioned in [FM00], which includes all the a-predictive a-states in which the predicted outputs appear as a contiguous prefix of the output sequence (note that our 1-predictive 1-state with its strong bias is not a fortuitous state, since we can predict the second output but not the first one). In fortuitous states, all the j's and $(S[i]+S[j])$'s in the first a rounds must be inside an interval of a consecutive known values (otherwise, these outputs can't be specified) and thus we can find these states via an exhaustive search on all the possible values of short segments in S.

Fluhrer and McGrew correctly stated that such an attack (see figure 3), based on fortuitous states, is impractical for the full version of RC4 with $n = 8$. However, they ignored the fact that it can lead to significant improvements in attacks on versions of RC4 with smaller values of n.

There are two possible benchmarks for the efficiency of such an attack: the time complexity and the number of required outputs. Denote the number of fortuitous states of order a by $F(a, N)$, and the probability that a (specific) fortuitous state happens as $pf(a, N) \approx N^{-(a+2)}$. The attack requires one fortuitous state to happen and thus the number of outputs needed for this attack is $\frac{1}{F(a,N)pf(a,N)}$. The time complexity of this attack is N (false candidates) times

```
FortAttack(Z^a, FDB^b)
    For each a-tuple ⟨x_1, ... , x_a⟩ of Z
        FList = Lookup(⟨x_1, ... , x_a⟩, FDB)
        If FList ≠ NULL
            For each state f ∈ FList
                Let S̄ be the partially guessed permutation according to f
                Let Z̄ be the next N words of Z
                S̄ = Complete^c(Z̄, S̄)
                If S̄ ≠ NULL
                    Return S̄
```

[a] the output stream
[b] table of fortuitous states
[c] Branch and Bound attack with apriori information

Fig. 3. A Fortuitous State Attack

the recursive function *complex* (defined in [KMP+98]), that describes the time complexity of the Branch and Bound attack with apriori information about a elements in S.

For small values of a the revealed part of S is insignificant, whereas for large values of a the output size increases dramatically. However, for RC4 with $n = 5$ this attack is feasible, and for other values of n it leads to considerable improvements in the time complexities of the best known attacks (see table 1)

		S									
i	j	-2	-1	0	1	2	3	$S[i]$	$S[j]$	$S[i]+S[j]$	$Output$
-3	-1	1	2	$*$	-1	$*$	$*$	/	/	/	/
-2	0	$*$	2	1	-1	$*$	$*$	$*$	1	$*$	$*$
-1	2	$*$	$*$	1	-1	2	$*$	$*$	2	$*$	$*$
0	3	$*$	$*$	$*$	-1	2	1	$*$	1	$*$	$*$
1	2	$*$	$*$	$*$	2	-1	1	2	-1	1	2
2	1	$*$	$*$	$*$	-1	2	1	2	-1	1	-1
3	2	$*$	$*$	$*$	-1	1	2	2	1	3	2

Fig. 4. A non-fortuitous 3-predictive 3-state

This attack can be further improved by finding predictive rather than fortuitous states. Every fortuitous state is predictive but the converse is clearly false, and thus we can get a better attack. However, we do not know how to estimate the approximate number of predictive states of various orders, and thus we cannot quantify the expected improvement. Small computational experiments had

Table 1. Performance of the fortuitous state attack (app. stands for approximation)

N	a	$F(a,N)$	$pf(a,N)$	Data	fp ratio	Time	Complex(0)
32	5	$2^{12.2}$	2^{-35}	$2^{22.8}$	$2^{4.5}$	$2^{41.7}$	2^{53}
	6	$2^{14.8}$	2^{-40}	$2^{25.2}$	$2^{4.3}$	$2^{38.6}$	
	7	2^{18}	2^{-45}	2^{27}	$2^{4.0}$	$2^{35.4}$	
	8	$2^{21.5}$	2^{-50}	$2^{28.5}$	$2^{3.6}$	$2^{32.3}$	
	9	(app.) $2^{24.5}$	2^{-55}	$2^{30.5}$	$2^{3.2}$	$2^{29.2}$	
64	5	$2^{12.9}$	2^{-42}	$2^{29.1}$	$2^{5.8}$	$2^{118.6}$	2^{132}
	6	$2^{15.2}$	2^{-48}	$2^{32.8}$	$2^{5.65}$	$2^{114.65}$	
	7	(app.)2^{17}	2^{-54}	2^{37}	$2^{5.5}$	$2^{110.7}$	
	8	(app.)2^{21}	2^{-60}	2^{39}	$2^{5.35}$	$2^{105.85}$	
	9	(app.)2^{24}	2^{-66}	2^{42}	$2^{5.2}$	2^{103}	
	10	(app.)2^{27}	2^{-72}	2^{45}	2^{5}	$2^{99.2}$	
128	4	$2^{11.7}$	2^{-42}	$2^{30.3}$	2^{7}	$2^{312.8}$	2^{324}
	8	(app.) 2^{24}	2^{-70}	2^{46}	2^{7}	$2^{287.5}$	
256	6	(app.) $2^{16.6}$	2^{-64}	$2^{47.4}$	2^{8}	$2^{755.2}$	2^{779}

yielded several interesting predictive states which are not fortuitous , such as the 3-state 3-predictive state described in figure 4, in which the predicted values appear surprisingly late in the generated sequence (at steps 4, 5, and 6).

		S									
i	j	1	2	3	4	5	6	7	8	9	10
0	-1	3	$*$	2	-1	$*$	$*$	$*$	$*$	$*$	$*$
1	2	$*$	3	2	-1	$*$	$*$	$*$	$*$	$*$	$*$
2	5	$*$	$*$	2	-1	3	$*$	$*$	$*$	$*$	$*$
3	7	$*$	$*$	$*$	-1	3	$*$	2	$*$	$*$	$*$
4	6	$*$	$*$	$*$	$*$	3	-1	2	$*$	$*$	$*$
5	9	$*$	$*$	$*$	$*$	$*$	-1	2	$*$	3	$*$
6	8	$*$	$*$	$*$	$*$	$*$	$*$	2	-1	3	$*$
7	10	$*$	$*$	$*$	$*$	$*$	$*$	$*$	-1	3	2
8	9	$*$	$*$	$*$	$*$	$*$	$*$	$*$	3	-1	2
9	8	$*$	$*$	$*$	$*$	$*$	$*$	$*$	-1	3	2
10	10	$*$	$*$	$*$	$*$	$*$	$*$	$*$	-1	3	2

Fig. 5. A 3-state that determines j for 10 rounds

Intuitively, a state that does not determine the values of j during the next r rounds, cannot predict the output word of the last of these rounds. In case j is "lost", it seems that it cannot be recovered because of the dependency of j on its former value. Thus, a predictive state must determine the values of j from its occurrence up to the last predicted round. A possible weakening of the notion

of predictive states is thus to require only that a prefix of j values should be uniquely determined by the a-state, without insisting on any predicted outputs. Despite the loss of our ability to filter false candidates, we earn the ability to track j values for d rounds conditioned by a constraints, where in some instances $d >> a$ (e.g., see figure 5).

5 Discussion

In sections 3 we described a significant statistical weakness of RC4, which had some interesting cryptanalytic consequences. However, we would like to stress that RC4 should not be considered as being completely broken, since our attack does not recover the key, and in many applications the ability to predict a single plaintext byte is of little importance. However, we believe that as a result of our observation, all future implementations of RC4 should skip at least the first two (and preferably the first N) output words.

References

[Fin94] H. Finney. an RC4 cycle that can't happen. September 1994.

[FM00] Fluhrer and McGrew. Statistical analysis of the alleged RC4 keystream generator. In *FSE: Fast Software Encryption*, 2000.

[Gol97] Golić. Linear statistical weakness of alleged RC4 keystream generator. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1997.

[Gol99] O. Goldreich. *Foundations of Cryptogtaphy.* 1 edition, 1999.

[GW00] A. L. Grosul and D. S. Wallach. a related-key cryptanalysis of RC4. June 2000.

[Jen] Robert J. Jenkins. ISAAC and RC4. Published on the internet at http://burtleburtle.net/bob/rand/isaac.html.

[KMP+98] Knudsen, Meier, Preneel, Rijmen, and Verdoolaege. Analysis methods for (alleged) RC4. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology.* LNCS, Springer-Verlag, 1998.

[MT98] Mister and Tavares. Cryptanalysis of RC4-like ciphers. In *SAC: Annual International Workshop on Selected Areas in Cryptography.* LNCS, 1998.

[Roo95] A. Roos. A class of weak keys in the RC4 stream cipher. September 1995.

[Sch96] B. Schneier. *Applied Cryptography.* John Wiley & Sons, Inc, Toronto, Canada, 2 edition, 1996.

Improved SQUARE Attacks against Reduced-Round HIEROCRYPT

Paulo S.L.M. Barreto[1], Vincent Rijmen[2⋆], Jorge Nakahara Jr.[2⋆⋆], Bart Preneel[2], Joos Vandewalle[2], and Hae Y. Kim[3]

[1] Scopus Tecnologia S. A.
Av. Mutinga, 4105 - Pirituba
BR–05110–000 São Paulo (SP), Brazil
pbarreto@scopus.com.br
[2] Katholieke Universiteit Leuven, Dept. ESAT,
Kasteelpark Arenberg 10,
B–3001 Leuven-Heverlee, Belgium
{vincent.rijmen,jorge.nakahara,bart.preneel,
joos.vandewalle}@esat.kuleuven.ac.be
[3] Universidade de São Paulo, Departamento de Engenharia Eletrônica.
Av. Prof. Luciano Gualberto, tr. 3, 158,
BR–05508-900, São Paulo (SP), Brazil.
hae@lps.usp.br

Abstract. We present improved SQUARE attacks against the NESSIE and ECTP candidate block ciphers HIEROCRYPT-3 and HIEROCRYPT-L1, designed by Toshiba. We improve over the previous best known attack on five S-box layers of HIEROCRYPT-3 by a factor of 2^{128} computational steps with an attack on six layers for 128-bit keys, and extend it to seven S-box layers for longer keys. For HIEROCRYPT-L1 we are able to improve previous attacks up to seven S-box layers (out of twelve).

1 Introduction

The HIEROCRYPT ciphers are candidate block ciphers for the NESSIE Project [8] and (ECTP) *Evaluation of Cryptographic Techniques Project* [3]. NESSIE is a project within the Information Societies Technology (IST) Programme of the European Commission (Key Action II, Action Line II.4.1). ECTP is part of the *MITI Action Plan for a Secure E-Government* announced by the Ministry of International Trade and Industry (MITI) of Japan in April 2000.

SQUARE attacks have first been described in [1]. Variations have been developed for related ciphers [10,2], and also for ciphers of quite distinct nature (as in [6], where the technique is called a 'saturation attack').

The designers of HIEROCRYPT-3 and HIEROCRYPT-L1 describe SQUARE-like attacks on reduced-round variants of the ciphers on the submission document.

⋆ F.W.O. Postdoctoral Researcher, sponsored by the Fund for Scientific Research – Flanders (Belgium)
⋆⋆ sponsored in part by GOA project Mefisto 2000/06

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 165–173, 2002.

The best attacks against HIEROCRYPT-3 [11, p. 8, Table 5] breaks four S-box layers (2 rounds) at the cost of 2^{40} key guesses using 2^{11} chosen plaintexts, and five S-box layers (2.5 rounds) at the cost of 2^{168} key guesses using 2^{13} chosen plaintexts. The best attack against HIEROCRYPT-L1 [12, p. 8, Table 4] breaks five S-box layers (2.5 rounds) at the cost of 2^{72} key guesses using 2^{32} chosen plaintexts. In this paper, we present more effective attacks on reduced-round variants of the HIEROCRYPT ciphers.

This paper is organised as follows. In Sect. 2 we review the structure of the HIEROCRYPT ciphers. Sect. 3 describes the principles behind the SQUARE attack. We describe our SQUARE-like attacks against HIEROCRYPT-3 in Sect. 4, and show how to apply them to HIEROCRYPT-L1 in Sect. 5. Sect. 6 provides some observations on an attack due to Gilbert and Minier against RIJNDAEL and its implications for HIEROCRYPT. Finally, Sect. 7 summarises the results.

2 Description of the HIEROCRYPT Ciphers

HIEROCRYPT-3 (HC-3) is an iterated 128-bit block cipher [13] with a variable number of rounds which depends on the cipher key size. For 128-bit keys, HC-3 has 6 rounds, for 192-bit keys there are 7 rounds, and for 256-bit keys, 8 rounds. The last round consists of an output transformation slightly different from the other rounds.

HC-3 has a hierarchical structure. At the highest level, an HC-3 round (see Fig. 1) consists of the following operations, in order:

- a layer of four simultaneous applications of 32×32-bit keyed substitution boxes (the XS-boxes).
- a diffusion layer consisting of a bytewise linear transform defined by the so-called MDS_H matrix.

Within each round, a similar structure exists. A 32×32-bit XS-box contains:

- a subkey mixing layer consisting of exoring the 32-bit input data with four subkey bytes. This layer is called the *upper* subkey layer in our attacks.
- a key-independent nonlinear layer composed of the parallel application of four 8×8-bit S-boxes. This layer is called *upper* S-box layer in our attacks.
- a diffusion layer consisting of a bytewise linear transform defined by the so-called MDS_L matrix, which is a Maximum Distance Separable (MDS) matrix [9,7].
- another subkey mixing layer consisting of exor with four subkey bytes. This layer is called *lower* subkey layer to contrast with the upper subkey layer.
- another key-independent nonlinear layer composed of the parallel application of four 8×8-bit S-boxes. Therefore, each *round* consists of *two* S-box layers. This layer is mentioned later as the *lower* S-box layer, to contrast with the previous similar layer.

The last round is an output transformation composed of an XS-box layer followed by an exor with the last round subkey (or alternatively, it has a round

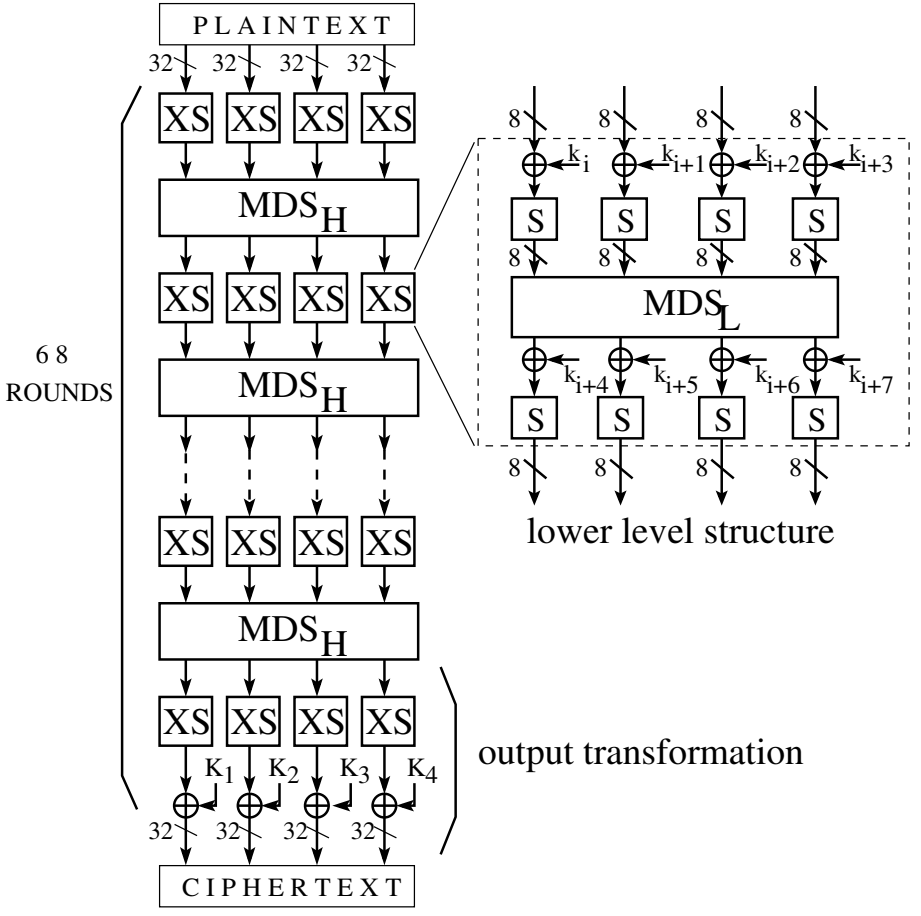


Fig. 1. Graphical representation of the HIEROCRYPT-3 encryption structure

structure with the MDS_H transform substituted by an exor layer with a subkey). The result is the 128-bit ciphertext block.

HIEROCRYPT-L1 (HC-L1) is a 64-bit block iterated cipher [14] using a 128-bit cipher key (see Fig. 2). It consists of six rounds plus an output transformation. Like HC-3, HC-L1 has a hierarchical structure. A high-level round consists of:

- a layer of two simultaneous applications of 32×32-bit keyed XS-boxes.
- a diffusion layer composed of a bytewise linear transform defined by the MDS'_H matrix.

Within each high-level round a similar structure exists. A 32-bit HC-L1 XS-box is identical to an HC-3 XS-box. The output transformation consists of an XS-box layer, followed by an exor mixing layer with the final 64-bit subkey. The result is the 64-bit ciphertext block.

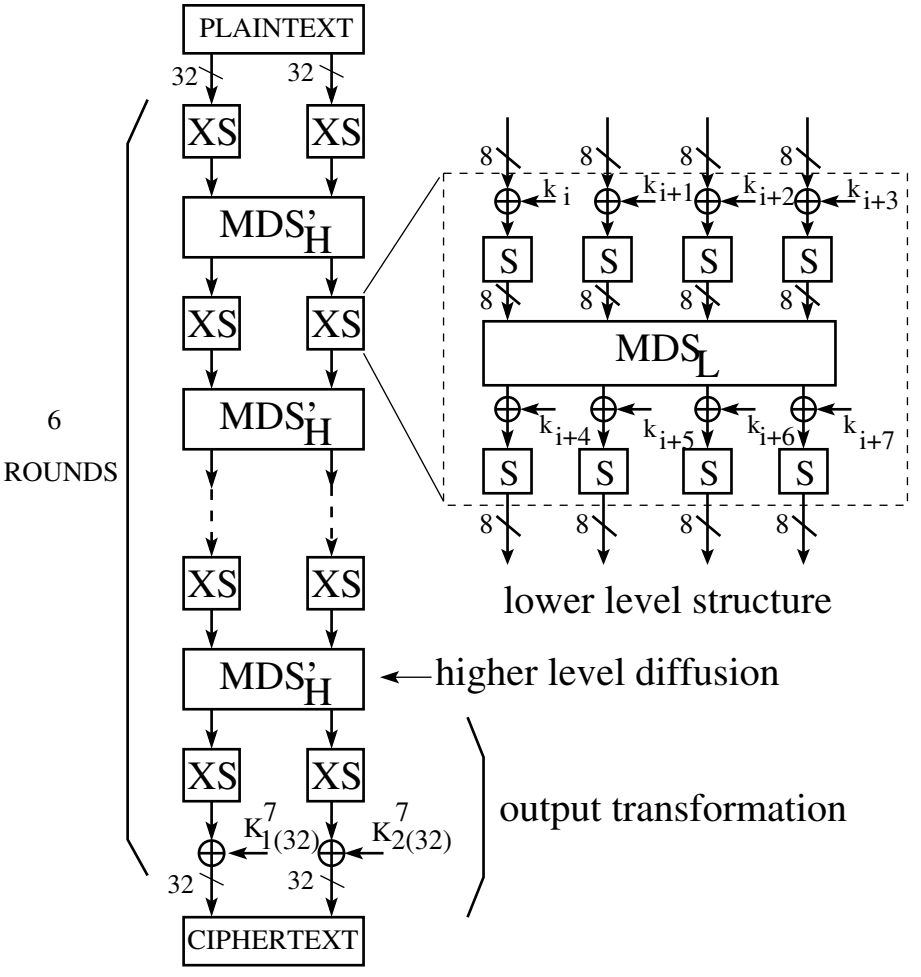


Fig. 2. Graphical representation of the HIEROCRYPT-L1 encryption structure

3 The SQUARE Attack

The SQUARE attack is a chosen-plaintext attack effective against reduced-round versions of block ciphers of the SQUARE family (see [1]). This attack explores the structure of SQUARE, where the *text* (the data being encrypted) is neatly partitioned into smaller component *words*. Let us provide some definitions.

Definition 1. *A method to distinguish the output of a certain number of rounds of an iterated cipher from the output of a truly random function is called a* distinguisher.

Definition 2. *([1]) A Λ-set is a set of* texts *that are all different in some of the component words (called* active words*) and are all equal in the other words (the* passive words*). Let $x = (x_1, \ldots, x_i, \ldots)$ and $y = (y_1, \ldots, y_j, \ldots)$ be two texts in the Λ-set, and λ be the set of indices of the active words. Then,*

$$\forall x, y \in \Lambda : \begin{cases} x_i \neq y_i \text{ for } i \in \lambda \\ x_i = y_i \text{ for } i \notin \lambda \end{cases}$$

The number of texts in a Λ-set depends on the word size. For SQUARE, where the word size is eight bits, a Λ-set consisted of 2^8 texts; in our present attacks a Λ-set consists of 2^{32} texts because we work with permutations of 32-bit words.

Definition 3. *(Balanced positions within a Λ-set) If the exor of all values at some word position in a Λ-set vanishes then that word position is called* balanced *over the Λ-set.*

The SQUARE attack starts by carefully choosing a Λ-set. By tracking the propagation of the active words through a round, it is possible to identify a pattern of active and passive words at the output of several rounds. That pattern contains a set of balanced words. These balanced words are used as a distinguishing test (further called the *Λ-distinguisher*) to identify the correct keys in the outer rounds of the cipher, either the first or the last round subkeys. Like SQUARE, HC-L1 and HC-3 possess a wordwise structure and similar variant attacks are possible.

The *partial sum technique* [4] is a dynamic programming technique that reduces the computational complexity of SQUARE attacks. It trades computational effort for storage by reorganising the intermediate computations. We refer to the original description of the technique [4] for a detailed explanation.

4 Attacking Reduced-Round HIEROCRYPT-3

4.1 An Observation on the Structure of the XS Transform

An equivalent representation of an XS-box, which is useful to describe our attack, can be obtained by pushing up the lower subkey layer to the input of the MDS_L matrix. This transformation creates an equivalent subkey layer.

Let the input to MDS_L be denoted by $X = (x_1, x_2, x_3, x_4) \in \mathrm{GF}(2^8)^4$ and its output by $Y = (y_1, y_2, y_3, y_4) \in \mathrm{GF}(2^8)^4$. Let $K = (k_1, k_2, k_3, k_4) \in \mathrm{GF}(2^8)^4$ be the subkey used in the lower subkey layer. The equivalent lower subkey layer is calculated by multiplying the original subkey by the MDS_L^{-1} transform:

$$MDS_L^{-1} = \begin{pmatrix} \mathsf{82}_x & \mathsf{C4}_x & \mathsf{34}_x & \mathsf{F6}_x \\ \mathsf{F6}_x & \mathsf{82}_x & \mathsf{C4}_x & \mathsf{34}_x \\ \mathsf{34}_x & \mathsf{F6}_x & \mathsf{82}_x & \mathsf{C4}_x \\ \mathsf{C4}_x & \mathsf{34}_x & \mathsf{F6}_x & \mathsf{82}_x \end{pmatrix}$$

For example, the equivalent most significant input to MDS_L is given by: $x_1 = \mathsf{82}_x \cdot (y_1 \oplus k_1) \oplus \mathsf{C4}_x \cdot (y_2 \oplus k_2) \oplus \mathsf{34}_x \cdot (y_3 \oplus k_3) \oplus \mathsf{F6}_x \cdot (y_4 \oplus k_4)$. It means, for instance, that the most significant subkey byte of the equivalent lower subkey layer is $ek_1 = \mathsf{82}_x \cdot k_1 \oplus \mathsf{C4}_x \cdot k_2 \oplus \mathsf{34}_x \cdot k_3 \oplus \mathsf{F6}_x \cdot k_4$.

4.2 Attacking Six S-Box Layers

We consider here a reduced-round version of HIEROCRYPT-3 that consists of two rounds, followed by the output transformation. In this version, there are three XS-box layers, which means six layers of the 8×8 S-box.

The main idea is to initially consider the nested structure as a black box and view the XS transform as the application of four keyed XS-boxes mapping $\mathrm{GF}(2^8)^4$ onto itself. The attack starts by choosing a set of 2^{32} plaintexts that have a fixed value for three of the four 32-bit blocks (the passive), while the fourth block (the active) takes all 2^{32} possible values. The position of the active block is not important. After the first round, all four 32-bit words are active (see [15], p. 5, Sect. 2.2.3).

After the second round the exor of all values from the Λ-set at each word position is zero (each word position is balanced). Since the round subkeys are constants (i.e. the same round subkeys are used to encrypt all the texts in the set), the words remain balanced after the addition with the upper round subkey of the third round (which is actually part of the output transformation). The attack proceeds by considering the ciphertexts. By guessing some bits of the round subkeys, we will partially decrypt up to the output of the second round and see whether the Λ-distinguisher holds. For correct guesses of the subkey bits, the Λ-distinguisher will always hold, while for wrong guesses the Λ-distinguisher will be destroyed with high probability. The following observation leads to a significant reduction of the attack complexity. Since the exor operation does not mix bits from different positions in the words, a 32-bit word being balanced implies all b-bit sub-blocks, $1 \leqslant b < 32$, will also be balanced. As a consequence, after the second round, every *byte* is balanced, and we only have to guess subkey bits to determine a single byte.

Now starting at the output, we have to guess 32 subkey bits that are added to one word in the output transformation, thus recovering the output from an XS-box. Looking at the inner structure of an XS-box, we see that we can invert the lower S-boxes as stated in Sect. 4.1, then move up the lower subkey layer, guess one byte of the equivalent subkey, undo the key addition and the upper S-box for one byte and verify the Λ-distinguisher.

Overall we are guessing 40 subkey bits. Since we are testing the Λ-distinguisher for one byte, we expect that one out of about 2^8 wrong keys will pass the test. If we repeat the attack with six Λ-sets, a wrong subkey will pass the test with probability $(2^{-8})^6 = 2^{-48}$. Since we try only 2^{40} subkeys, it is likely that only the correct one is able to pass through.

At first sight, it would seem that the 2^{40} key guessing steps should be repeated for each of the 2^{32} plaintexts in a Λ-set, with a resulting complexity of 2^{72} S-box lookups per Λ-set. However, the partial sum technique [4, section 2.3] provides a more efficient way to organise the guessing. At the modest cost of about 2^{24} extra bits, it reduces the total attack complexity to only 5×2^{48} S-box lookups per Λ-set.

Therefore, the complexity of the attack on six S-box layers is 6×2^{32} chosen plaintexts (six Λ-sets), and $6 \times 5 \times 2^{48} \approx 2^{53}$ S-box lookups.

4.3 Attacking Seven S-Box Layers

Attacking seven S-box layers is effective only for 192-bit and 256-bit keys. The strategy consists of guessing the first subkey (128 key bits) layer in an XS-box, and just use the previous attack on six S-box layers.

The computational effort becomes $2^{128+40} = 2^{168}$ subkey guesses. As 2^8 wrong subkeys will be filtered out per guess, and we guess 168 subkey bits at a time, the number of chosen plaintexts required to assure that only the correct subkey survives the filtering is $(\frac{168}{8}+1) \times 2^{32} = 22 \times 2^{32}$. By applying the partial sum technique, the overall complexity becomes $22 \times 5 \times 2^{48+128} \approx 2^{183}$ S-box lookups.

5 Attacking Reduced-Round HIEROCRYPT-L1

5.1 Attacking Six S-Box Layers

The best published attack against HIEROCRYPT-L1 is due to the cipher designers (see [12], p. 8, Table 4) and breaks five S-box layers at the cost of 2^{72} subkey guesses using 2^{32} chosen plaintexts. The same attack described in Sect. 4.2 for HIEROCRYPT-3 works against six S-box layers of HIEROCRYPT-L1, with the same requirements of $\approx 2^{53}$ S-box lookups and 6×2^{32} chosen plaintexts.

5.2 Attacking Seven S-Box Layers

The strategy for attacking seven S-box layers consists of guessing the complete last subkey (64 key bits) and using the previous attack on six S-box layers.

The computational effort becomes $2^{64+40} = 2^{104}$ subkey guesses. As 2^8 wrong subkeys will be filtered out per guess, and we guess 104 subkey bits at a time, the number of chosen plaintexts required to ensure that only the correct subkey survives the filtering is $(\frac{104}{8}+1) \times 2^{32} = 14 \times 2^{32}$. By applying the partial sum technique, the overall complexity becomes $14 \times 5 \times 2^{48+64} \approx 2^{118}$ S-box lookups.

6 The Gilbert-Minier Attack

The Gilbert-Minier attack [5], like the SQUARE attack, is based on distinguisher. In the terminology of [5], the SQUARE attack is based on a 3-round distinguisher. By guessing some round key bytes, it is possible to break six rounds of SQUARE. Gilbert and Minier developed a 4-round distinguisher for SQUARE (or RIJNDAEL), which makes it possible to break seven rounds.

In the case of the HIEROCRYPT ciphers, our extension of the SQUARE attack is already based on a 4-layer distinguisher. An improvement would thus require the construction of a 5-layer distinguisher. According to our analysis, the alternation of MDS_L and MDS_H (or MDS'_H) effectively prohibits the construction of a 5-round distinguisher. However, we feel that this issue deserves further investigation.

7 Conclusion

We presented improved SQUARE-like attacks against reduce-round versions of HIEROCRYPT-3, for six and seven S-box layers (3 and 3.5 rounds), compared to previous figures reported by its designers. The new attack requirements are summarised in Table 1. The columns labeled "subkey guesses," "S-box lookups," and "encryptions" actually refer to the same attack effort measured in different units. For the rightmost column (attack effort measured in number of encryptions) we use the approximate equivalence 1 encryption $\approx 2^7$ S-box lookups.

Table 1. Attack requirements for HIEROCRYPT-3

Attack	S-box layers	chosen plaintexts	subkey guesses	S-box lookups	encryptions
HC-3 paper[11]	4	2^{11}	2^{40}	$\leqslant 2^{51}$	$\leqslant 2^{44}$
HC-3 paper[11]	5	2^{13}	2^{168}	$\leqslant 2^{181}$	$\leqslant 2^{174}$
ours	6	6×2^{32}	2^{40}	2^{53}	2^{46}
ours	7	22×2^{32}	2^{168}	2^{183}	2^{176}

Table 2. Attack requirements for HIEROCRYPT-L1

Attack	S-box layers	chosen plaintexts	subkey guesses	S-box lookups	encryptions
HC-L1 paper[12]	5	2^{32}	2^{72}	$\leqslant 2^{104}$	$\leqslant 2^{97}$
ours	6	6×2^{32}	2^{40}	2^{53}	2^{46}
ours	7	14×2^{32}	2^{104}	2^{118}	2^{111}

The improved attacks work also for HIEROCRYPT-L1, with the appropriate changes due to the block size. The new attack requirements are summarised in Table 2. Again, the columns labeled "subkey guesses," "S-box lookups," and "encryptions" actually refer to the same attack effort measured in different units, and for the rightmost column we use the approximate equivalence 1 encryption $\approx 2^7$ S-box lookups. Our attacks, nonetheless, do not represent a threat to the security of either cipher.

References

1. J. Daemen, L.R. Knudsen, V. Rijmen, "The Block Cipher SQUARE," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 149–165.
2. C. D'Halluin, G. Bijnens, V. Rijmen, B. Preneel, "Attack on Six Rounds of CRYPTON," *Fast Software Encryption, LNCS 1636*, L. Knudsen, Ed., Springer-Verlag, 1999, pp. 46–59.

3. "Evaluation of Cryptographic Techniques Project," http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html.
4. N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved Cryptanalysis of RIJNDAEL," to appear in *Fast Software Encryption'00*, Springer-Verlag.
5. H. Gilbert, M. Minier, "A Collision Attack on Seven Rounds of RIJNDAEL," *Third Advanced Encryption Standard Candidate Conference*, NIST, April 2000, pp. 230–241.
6. S. Lucks, *"The Saturation Attack – A Bait for Twofish,"* these Proceedings.
7. F.J. MacWilliams, N.J.A. Sloane, "The Theory of Error-Correcting Codes," *North-Holland Mathematical Library*, vol. 16, 1977.
8. NESSIE Project – New European Schemes for Signatures, Integrity and Encryption – http://cryptonessie.org.
9. V. Rijmen, "Cryptanalysis and Design of Iterated Block Ciphers," *Doctoral Dissertation*, October 1997, K.U.Leuven.
10. V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, E. De Win, "The Cipher SHARK," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 99-112
11. Toshiba Corporation, "Security Evaluation: HIEROCRYPT-3," September 25, 2000 – available at http://cryptonessie.org.
12. Toshiba Corporation, "Security Evaluation: HIEROCRYPT-L1," September 25, 2000 – available at http://cryptonessie.org.
13. Toshiba Corporation, "Specification of HIEROCRYPT-3," submitted to the First Open NESSIE Workshop, 13-14 November 2000, Leuven, Belgium – available at http://cryptonessie.org.
14. Toshiba Corporation, "Specification of HIEROCRYPT-L1," submitted to the First Open NESSIE Workshop, 13-14 November 2000, Leuven, Belgium – available at http://cryptonessie.org.
15. Toshiba Corporation, "Specification on a Block Cipher: HIEROCRYPT-3," Toshiba Corporation, Sep. 15, 2000 – submitted to the First Open NESSIE Workshop, 13-14 November 2000, Leuven, Belgium – available at http://cryptonessie.org.

Differential Cryptanalysis of Q*

Eli Biham[1], Vladimir Furman[1], Michal Misztal[2], and Vincent Rijmen[3]**

[1] Computer Science Department, Technion – Israel Institute of Technology, Haifa 32000, Israel. {biham,vfurman}@cs.technion.ac.il, http://www.cs.technion.ac.il/~biham/.
[2] Institute of Mathematics and Operational Research Military University of Technology, 00-908 Warsaw, ul. Kaliskiego 2, Poland
[3] ESAT/COSIC, K.U.Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium

Abstract. Q is a block cipher based on Rijndael and Serpent, which was submitted as a candidate to the NESSIE project by Leslie McBride. The submission document of Q describes 12 one-round iterative characteristics with probability 2^{-18} each. On 7 rounds these characteristics have probability 2^{-126}, and the author of Q claims that these are the best 7-round characteristics. We find additional one-round characteristics that can be extended to more rounds. We also combine the characteristics into differentials. We present several differential attacks on the full cipher. Our best attack on the full Q with 128-bit keys (8 rounds) uses 2^{105} chosen plaintexts and has a complexity of 2^{77} encryptions. Our best attack on the full Q with larger key sizes (9 rounds) uses 2^{125} chosen ciphertexts, and has a complexity of 2^{96} for 192-bit keys, and 2^{128} for 256-bit keys.

1 Introduction

Q is a block cipher submitted as a candidate to the NESSIE project [6] by Leslie McBride [5]. The best previous differential attack on Q is presented in its submission document (see [2] for more details about differential cryptanalysis). The documentation of Q describes 12 one-round iterative differential characteristics with probability 2^{-18} each. The author of Q claims that the best 7-round characteristics of Q are constructed by concatenating these one-round characteristics. These 7-round characteristics have probability 2^{-126}.

In this paper we show additional one-round characteristics that can be extended to more rounds. We also join the characteristics into differentials [4]. Subsequently, we present several differential attacks on full Q. A straightforward attack on 8 rounds uses 2^{124} chosen plaintexts and has negligible complexity. A more advanced attack on 8 rounds uses 2^{105} chosen plaintexts and has

* The work described in this paper has been supported by the European Commission through the IST Programme under Contract IST-1999-12324 and by the fund for the promotion of research at the Technion.

** FWO research assistant, sponsored by the Fund for Scientific Research - Flanders (Belgium). This research was sponsored in part by GOA project Mefisto 2000/06.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 174–186, 2002.

a complexity of 2^{77}. Our advanced attack on 9 rounds uses about 2^{125} chosen ciphertexts and has a complexity of 2^{128}. We discuss how to improve the complexity of this attack to 2^{96} in the case of 192-bit keys. In the 8- and 9-round attacks, we can reduce the complexity by using more plaintexts/ciphertexts, or reduce the number of chosen plaintexts/ciphertexts with a larger complexity of analysis.

The paper is organized as follows: In Section 2 we describe the structure of Q. In Section 3 we show the characteristics and the differentials of Q, for various numbers of rounds. In Section 4 we present the straightforward attack. In Section 5 we describe the more advanced attacks. We present a short conclusion in Section 6. Finally, in Appendix A we show the key scheduling of Q.

2 The Structure of Q

The cipher has 128-bit blocks and the key size may be 128 bits, 192 bits or 256 bits. The block is divided into four 32-bit words (a word consists of four bytes):

word 0	word 1	word 2	word 3
03	13	23	33
02	12	22	32
01	11	21	31
00	10	20	30

where '00' is the least significant byte, and '33' is the most significant byte. Groups of four bytes taken from different words, but from the same position in the words, are called *rows* (for example, the group of bytes '03','13','23' and '33' is a row).

Figure 1 describes the round function of Q. The Keying layers XOR the corresponding subkeys (KA, KB or K_n) to the data, where KA and KB are fixed in all rounds, and K_r is a subkey used in round r only. The Byte Substitution layer is taken from Rijndael [3]. It substitutes the value of each byte in the data according to the S-Table. The Bit-Slice layer is taken from Serpent [1]: for $i = 0, \ldots, 31$, we construct nibble i by taking bit i from every word, then we replace each nibble according to the S-box (either A or B) and return the new bit values to their original places. The permutation changes the order of the bytes in the words in the following way: word 0 is not changed, word 1 is rotated by 1 byte (the bytes '10','11','12','13' become '13','10','11','12'), word 2 is rotated by 2 bytes and word 3 is rotated by 3 bytes. To simplify the following analysis we divide the round operations into two parts, called 'part I' and 'part II', as shown in Figure 1.

The full cipher consists of 8 rounds in the case of 128-bit keys and 9 rounds for longer key lengths. The full cipher is illustrated in Figure 2.

3 Differentials of Q

Figure 3 describes the one-round characteristic with probability 2^{-18} presented

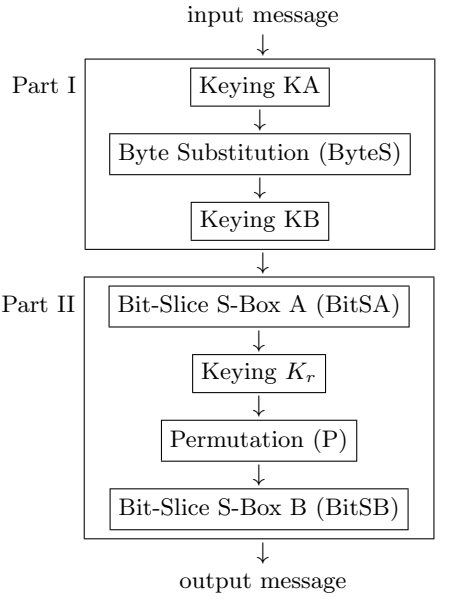


Fig. 1. The round function of Q

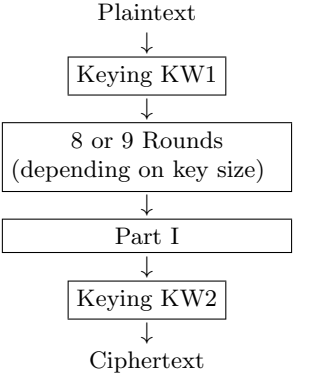


Fig. 2. Outline of Q

$$\begin{bmatrix} 0 & \delta_i & 0 & \delta_i \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow[ByteS]{2^{-14}} \begin{bmatrix} 0 & \delta_i & 0 & \delta_i \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow[BitSA]{2^{-2}} \begin{bmatrix} 0 & \delta_i & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\xrightarrow[P]{1} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \delta_i & 0 & 0 \end{bmatrix} \xrightarrow[BitSB]{2^{-2}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \delta_i & 0 & \delta_i \end{bmatrix}$$

Fig. 3. The one-round characteristic described by McBride

by McBride in [5], where δ_i is the 8-bit value 2^i ($i \in \{0, \ldots, 7\}$). The characteristic holds when $i \in \{0, 5, 7\}$. The input differences δ_i can be in any row. The row in which the output difference occurs will vary accordingly. Hence, in total $3 \times 4 = 12$ similar one-round characteristics were presented.

In the following we present additional one-round characteristics and expand the characteristics to one-round differentials.

3.1 One-Round Differentials

For simplicity of analysis we initially ignore the permutation layer, and present the analysis on a version of Q without this layer. Later, we will adapt the received results to the original Q, and discuss the influence of the permutation layer on the analysis. In our analysis, bits, bytes and words with non-zero difference will be called *active* bits, bytes or words, respectively.

The additional one-round characteristics that we present are similar to the characteristics presented by McBride. In Part I, McBride uses only characteristics where the active bits of the input difference and the active bits of the output difference are the same. We, in contrast, allow that the active bits of input and of output differences are not the same. Figure 4 describes the characteristic of

$$\begin{bmatrix} 0 & \delta_i & 0 & \delta_i \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \overset{\substack{2^{-14} \\ \rightarrow \\ Part\ I}}{} \begin{bmatrix} 0 & \delta_j & 0 & \delta_j \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Fig. 4. The characteristic of Part I

Part I. In McBride's characteristics i always equals j, while in our characteristics this constraint is removed.

Table 1 shows the probabilities for all 1-bit input differences to cause 1-bit output differences in the 8-bit S-box S that is used in the Byte Substitution layer. All empty entries have probability 0. In the characteristics presented in [5], only the entries laying on the diagonal are used. Our analysis uses all the non-zero entries. The characteristic over Part I uses the same entry of Table 1 twice, so we convert this table to the Part I table by squaring the probability of every non-zero entry. The resulting matrix is denoted by M_I.

In Part II, McBride uses only the characteristic where an input difference that has two active bits in one nibble (bits 1 and 3) causes a difference with one active bit (bit 1) after the Bit-Slice A layer, and an output difference with two active bits in one nibble (again bits 1 and 3). In the following, we describe additional characteristics over Part II, where an input difference with two active bits in one nibble causes an output difference with two active bits in one nibble. The active bits in the output difference are not necessarily the same bits as in the input difference.

Table 1. 8-bit S-box S: The probability that an input difference with exactly one active bit (i) causes an output difference with exactly one active bit (j).

i	j							
	0	1	2	3	4	5	6	7
0	$\mathbf{2^{-7}}$		2^{-7}	2^{-7}				2^{-7}
1			2^{-7}				2^{-7}	2^{-7}
2						2^{-7}		
3		2^{-7}			2^{-7}			
4	2^{-7}			2^{-7}		2^{-7}		
5		2^{-7}	2^{-7}			$\mathbf{2^{-7}}$	2^{-7}	2^{-7}
6	2^{-7}				2^{-7}	2^{-7}		
7		2^{-7}		2^{-7}				$\mathbf{2^{-7}}$

Table 2. 4-bit S-box A: The probabilities that input differences with two active bits cause output differences with exactly one active bit.

M_A :

active input bits	active output bit 0	1	2	3
0,1	2^{-3}			2^{-2}
0,2				2^{-2}
1,2	2^{-3}			
0,3		2^{-2}	2^{-3}	
1,3		$\mathbf{2^{-2}}$	2^{-3}	
2,3	2^{-3}	2^{-2}	2^{-3}	

Table 2 describes the probabilities that input differences with two active bits cause output differences with exactly one active bit, for the 4-bit S-box A. The empty entries denote zero probability. After the Bit-Slice A layer the difference has one active bit, so the input difference of the Bit-Slice B layer also has one active bit.

Table 3. 4-bit S-box B: The probabilities that input differences with one active bit cause output differences with two active bits.

M_B :

active input bit	active output bits 0,1	0,2	1,2	0,3	1,3	2,3
0	2^{-3}	2^{-2}	2^{-3}			2^{-3}
1		2^{-3}		2^{-3}	$\mathbf{2^{-2}}$	
2			2^{-2}		2^{-3}	2^{-3}
3	2^{-3}		2^{-3}	2^{-3}		2^{-3}

Table 3 describes the probabilities that input differences with one active bit cause output differences with two active bits, for the 4-bit S-box B. McBride uses only the entries that are in bold in both tables (Table 2 and Table 3).

Table 4. Part II: The probabilities that input differences with two active bits cause output differences with two active bits.

$M_{II} = M_A \cdot M_B$:

active	active output bits					
input bits	0,1	0,2	1,2	0,3	1,3	2,3
0,1	$3 \cdot 2^{-6}$	$2 \cdot 2^{-6}$	$3 \cdot 2^{-6}$	$2 \cdot 2^{-6}$		$3 \cdot 2^{-6}$
0,2	$2 \cdot 2^{-6}$		$2 \cdot 2^{-6}$	$2 \cdot 2^{-6}$		$2 \cdot 2^{-6}$
1,2	$1 \cdot 2^{-6}$	$2 \cdot 2^{-6}$	$1 \cdot 2^{-6}$			$1 \cdot 2^{-6}$
0,3		$2 \cdot 2^{-6}$	$2 \cdot 2^{-6}$	$2 \cdot 2^{-6}$	$5 \cdot 2^{-6}$	$1 \cdot 2^{-6}$
1,3		$2 \cdot 2^{-6}$	$2 \cdot 2^{-6}$	$2 \cdot 2^{-6}$	$5 \cdot 2^{-6}$	$1 \cdot 2^{-6}$
2,3	$1 \cdot 2^{-6}$	$4 \cdot 2^{-6}$	$3 \cdot 2^{-6}$	$2 \cdot 2^{-6}$	$5 \cdot 2^{-6}$	$2 \cdot 2^{-6}$

Table 4 is the product of the matrixes M_A and M_B, $M_{II} = M_A \cdot M_B$. It describes the probabilities that input differences with two active bits cause output differences with two active bits in Part II. The empty entries denote zero probability. The entry (1,3),(1,3), for example, has probability $5 \cdot 2^{-6}$, because in addition to the McBride's characteristic described earlier we have the characteristic that goes $(1,3) \rightarrow (2) \rightarrow (1,3)$ and has probability 2^{-6}.

Now we extend the one-round characteristics to one-round differentials. To build a one-round differential, we compute the tensor product of M_I and M_{II}: $M_R = M_I \cdot M_{II}$. The matrix M_R contains the probabilities of corresponding one-round differentials, where every such differential may start from any row (i.e., we take any non-zero entry from Table 1 and any non-zero entry from Table 4). We obtain $24 \cdot 29 \cdot 4 = 2784$ one-round differentials (24 non-zero entries in Table 1, 29 non-zero entries in Table 4 and 4 options for the starting row). The best one-round differentials have probability $2^{-14} \cdot 5 \cdot 2^{-6} = 5 \cdot 2^{-20}$. One of them is shown on Figure 5 where i does not necessarily equal j.

$$\begin{bmatrix} 0 & \delta_i & 0 & \delta_i \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \underset{\text{Part I}}{\overset{2^{-14}}{\rightarrow}} \begin{bmatrix} 0 & \delta_j & 0 & \delta_j \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \underset{\text{Part II}}{\overset{5 \cdot 2^{-6}}{\rightarrow}} \begin{bmatrix} 0 & \delta_j & 0 & \delta_j \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Fig. 5. One-round differential with probability $5 \cdot 2^{-20}$

3.2 Differentials over Several Rounds of Q

In this subsection we extend the one-round differentials to more rounds. For this analysis we assume that the rounds act independently, such that the probability of a multiple-round characteristic can be approximated by the product of the probabilities of one-round characteristics.

Part I changes only the index of active bits in the two bytes (of course, it must change to the same bit in both bytes), but does not change which bytes

are active. Part II changes only which bytes in the row are active, but does not change the active row, nor the index of the active bit inside the bytes. Hence, the differentials of Part I over several rounds and the differentials of Part II over several rounds may be built independently, by making sure that both are satisfied together.

Thus, in order to obtain the probability of differentials on i rounds, we compute $(M_R)^i = (M_I)^i \times (M_{II})^i$. For example, Table 5 shows $(M_I)^2$ and $(M_{II})^2$.

Table 5. The matrixes M_I and M_{II} for 2-round differentials

$$(M_I)^2 = (2^{-14})^2 \cdot \begin{pmatrix} 1&2&1&2&1&1&0&2 \\ 1&1&0&1&1&2&0&1 \\ 0&1&1&0&0&1&1&1 \\ 1&0&1&1&0&0&2&1 \\ 2&1&1&1&2&1&0&1 \\ 1&2&2&1&1&3&2&3 \\ 2&1&2&2&0&1&2&2 \\ 0&2&1&1&1&0&1&2 \end{pmatrix}$$

$$(M_{II})^2 = (2^{-6})^2 \cdot \begin{pmatrix} 19&28&29&20&25&24 \\ 10&20&18&12&20&14 \\ 9&8&11&8&5&10 \\ 7&22&23&20&40&15 \\ 7&22&23&20&40&15 \\ 16&30&34&28&45&25 \end{pmatrix}$$

Consulting Table 5, we see that the best probability of two-round differential is $3 \cdot (2^{-14})^2 \cdot 45 \cdot (2^{-6})^2 = 135 \cdot 2^{-40} \cong 2^{-33}$.

In this way, we can calculate the probabilities of the differentials over several rounds up to 9 rounds. We can also calculate the probabilities of the differentials over several rounds with an additional Part I in the end, or with an additional Part II in the beginning. For every such case there are $64 \cdot 36 \cdot 4 = 9216$ differentials (the computation is similar to the computation for the one-round case described earlier). Table 6 presents the best probabilities of the differentials for 6 rounds

Table 6. The best differential probabilities of Q (without permutation layer) for various numbers of rounds

Number of rounds	Normal case	With additional Part I	With additional Part II
6	$2^{-92.9}$	$2^{-105.35}$	$2^{-95.5}$
7	$2^{-107.9}$	$2^{-120.35}$	$2^{-110.5}$
8	$2^{-122.9}$	$2^{-135.35}$	$2^{-125.5}$
9	$2^{-137.9}$	$2^{-150.35}$	Irrelevant

and more. The best differentials are obtained when we start with δ_5 in bytes of the same row in words 2 and 3 (for example, in bytes '23' and '33'), and end with δ_7 in bytes of the same row in words 1 and 3. For the case that the matrix M_I is used j times and the matrix M_{II} is used k times, the appropriate entries are (5,7) in the $(M_I)^j$, and ((2,3),(1,3)) in $(M_{II})^k$. The probabilities of the other (not the best) differentials are close to the probability of the appropriate best differential presented in Table 6.

Now we discuss how the permutation layer influences the results. In the presented differentials, the permutation layer always works with one active byte only, and this layer only changes the order of the bytes. So, the permutation layer does not change the index of the active bit (i.e., has no influence on Part I). It also has no influence on Part II, because the permutation layer does not change the active word. Only the final row of the differential is influenced. In the variant of Q without a permutation layer, every differential has the active bits of the output difference in the same row as the active bits of the input difference (the difference cannot be in another row). In the original Q, however, different characteristics have the active bits of the output difference in different rows. These characteristics can be joined to 4 differentials (one for every possible active final row), where the sum of their probabilities equals the probability of the single differential that we obtain on the variant of Q without permutation.

Note that all the differentials described in this section may be taken in the backward direction. The probabilities of the differentials in the backward direction are equal to the presented probabilities for the forward direction.

4 A Basic Attack on Q

In this attack, we use a differential that spans the first 7 rounds, and part I and Bit-Slice A of the 8th round. In the first round, we can increase the probability by choosing input differences that produce the correct output difference with a higher probability than in the differentials presented in Section 3. Indeed, there exist differences ϵ such that the probability that the input difference ϵ may cause the output difference δ_i is higher than the probability that δ_j may cause δ_i. The optimal input difference is $\epsilon =$ `0xd8`. In the 8th round, we do not specify the position of the active bits within the active bytes and consider all these possibilities as a generalized kind of differential, which may have different values for δ_i in the last round. The probability of the generalized differential is 2^{-121}. It has the following input difference:

$$\begin{bmatrix} 0\,0 & 0 & \texttt{0xd8} & \texttt{0xd8} \\ 0\,0 & 0 & 0 & 0 \\ 0\,0 & 0 & 0 & 0 \\ 0\,0 & 0 & 0 & 0 \end{bmatrix},$$

and after Bit-Slice A in the 8th round, it has one active bit, but we do not know the position of that bit. In order to simplify the attack, we specify the active

row in the output. This reduces the probability of the differential by a factor of 1/4 (approximately).

Pairs that follow this differential have one active bit at the output of Bit-Slice A in the last round, hence one active row at the output of the 8th round, and hence one active row in the ciphertext. A pair that does not follow the characteristic has a similar output difference with probability 2^{-96}. Every pair that is not filtered suggests about 8 values for one word of $KW2 \oplus KB$. The signal-to-noise ratio of this attack is then:

$$S/N = \frac{2^{32} \cdot 2^{-123}}{8 \cdot 2^{-96}} = 4 \ .$$

The attack uses 2^{32} counters to count on 32 key bits. Since the S/N ratio is larger than 1, a counter with more than 4 hits is likely to belong to the correct key word. In order to get 4 right pairs, 2^{124} chosen plaintexts have to be encrypted. The plaintext requirements can be reduced by using structures of pairs. However, the attack discussed in the next section uses significantly fewer plaintexts.

5 Optimized Attacks on Q

Let G be the set of differences with exactly two active bytes, where both bytes are in the same row, and both have the same differential δ_i for any $i \in \{0, \ldots, 7\}$. In total, there are $\binom{4}{2} \cdot 4 \cdot 8 = 192$ differences in G (i.e., $\binom{4}{2}$ for choosing the affected bytes in the row, 4 for choosing the row and 8 for choosing i).

5.1 128-Bit Keys

In the case of 128-bit keys, the cipher consists of 8 rounds. We recover the key using about 2^{105} chosen plaintexts and 2^{77} complexity.

In this attack, we use the set of 7-round differentials described earlier, where the input difference is in the set of

$$\begin{bmatrix} 0 & 0 & \delta_i & \delta_i \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

for all $i \in \{0, \ldots, 7\}$, as described in Section 3.2, and the output difference is in the set G. The sum of probabilities of the differentials in this set with any fixed input difference is approximately $2^{-104.35}$. So given about 2^{107} pairs with such an input difference we get an expected number of 6 pairs with output differences that belong to G. We use this set of 7-round differentials starting with Part II of the first round. The characteristic extends until the start of Part II in the last round.

We choose a structure of 2^{16} plaintexts by taking all the possibilities of bytes indexed '23' and '33', and all the other bytes are fixed to some randomly selected

value. This structure includes 2^{31} pairs. The plaintexts give all the possible values of bytes '23' and '33' after Part I, and thus, they give $8 \cdot 2^{15} = 2^{18}$ pairs with the required difference (for the 7-round differential) after Part I. Hence, we need about $2^{107}/2^{18} = 2^{89}$ such structures in order to get an expected number of 6 pairs following the characteristic, i.e., we need about $2^{89} \cdot 2^{16} = 2^{105}$ chosen plaintexts. We call a pair with a difference that belongs to G after 7.5 rounds a *right pair*. We request to encrypt these 2^{105} chosen plaintexts to their ciphertexts under the unknown key.

How do we recognize the right pairs? Every right pair has difference δ_i (for some $i \in \{0, \ldots, 7\}$) in two bytes of the same row after 7.5 rounds, and zero differences in all other bytes. Different right pairs may have different i's. After an additional Part II, every right pair has either difference δ_i or 0 in every byte, but at least one row includes at least two bytes with difference δ_i. The received values become ciphertexts after an additional Part I. An input difference δ_i (for any fixed i) can result in approximately 128 possible output differences after S-box S. The set of possible output differences resulting from input difference δ_i is denoted by Z_i. Due to the fact that all the byte differences in the received values in the right pairs are either δ_i or 0, every byte of their ciphertext differences must either belong to Z_i or be 0 respectively. We compute Z_i for all $i \in \{0, \ldots, 7\}$ (this can be performed in a preprocessing stage). We check whether the output differences of all the bytes belong to the same Z_i. If they do, we continue the analysis. Otherwise this pair cannot be a right pair, and we discard it.

Wrong pairs satisfy the above condition for some specific Z_i with probability $(1/2)^{16} = 2^{-16}$ (1/2 is the probability that one byte difference belongs to this Z_i and 16 is the number of bytes). We select the pairs that satisfy the above condition for some Z_i. Thus about $2^{89} \cdot 2^{31} \cdot 2^{-16} \cdot 8 = 2^{107}$ pairs are selected.

Every selected pair suggests about 2^{16} possible values for $KW2 \oplus KB$. The 6 right pairs must suggest the right value of $KW2 \oplus KB$. So we look for values that are suggested 6 or more times. In total, about $2^{16} \cdot 2^{107} = 2^{123}$ values are suggested. We expect that approximately $\binom{2^{123}}{6} \cdot (2^{-128})^5 \cong 2^{89}$ values are suggested by 6 or more pairs.

The 6 right pairs must also suggest the right value of 16 bits of $KW1 \oplus KA$ (bytes '23' and '33'). Every selected pair suggests about $8^2 = 64$ possible such 16-bit values. So for every selected group of 6 pairs the probability to have at least one value suggested by all 6 pairs is $64^6 \cdot (2^{-16})^5 = 2^{-44}$. Thus, only $2^{89} \cdot 2^{-44} = 2^{45}$ groups of 6 pairs remain.

We observe that the subkey KB has only 2^{32} possible values (see Appendix A). For each guessed KB, we find $KW2$ by XOR-ing this KB with each of the 2^{45} combinations received from the 2^{45} groups. According to the key scheduling algorithm described in Appendix A, we can recover all the subkeys and the key given $KW2$ in an amount of time that is equivalent to the time taken to perform a single encryption. The wrong keys can be discarded by comparing the guessed KB with KB received from $KW2$. If more than one key remains after this check, then we discard all the remaining wrong keys by trial encryption.

The attack requires 2^{105} chosen plaintexts, and the complexity of analysis is about $2^{45} \cdot 2^{32} = 2^{77}$.

There is also a similar *chosen ciphertext attack* which requires the same number of chosen ciphertexts and has the same complexity.

5.2 256-Bit and 192-Bit Keys

In this case, the cipher consists of 9 rounds. We recover the key using about 2^{125} chosen ciphertexts and 2^{128} complexity.

The attack is similar to the attack from Section 5.1, except that this attack operates in the backward direction. According to Section 3.2 the best differentials in the backward direction start with active bytes in words 1 and 3. Thus, in this attack, we use the set of 8-round differentials in the backward direction, where the input difference is in the set of

$$\begin{bmatrix} 0 & \delta_i & 0 & \delta_i \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

for all $i \in \{0, \ldots, 7\}$, and the output difference belongs to G. The sum of probabilities of the differentials in this set with any fixed input difference is approximately $2^{-119.35}$. So given about 2^{127} pairs with such an input difference we expect to have about 200 pairs ($200 \cong 2^{7.65}$) with output differences that belong to G. We use this set of 8-round differentials starting with Part II of the 9th round.

We build structures of 2^{16} ciphertexts each, as in the previous attack, but now we take all possibilities for bytes '13' and '33'. In this attack we need about $2^{127}/2^{18} = 2^{109}$ such structures to get an expected number of 200 right pairs after 8.5 rounds in the backward direction (or after the first round in the forward direction), i.e., we need about $2^{109} \cdot 2^{16} = 2^{125}$ chosen ciphertexts. Now we call a pair with difference that belongs to G after 8.5 rounds a *right pair*. We request to decrypt these 2^{125} chosen ciphertexts to their plaintexts under the unknown key.

The pair selection process is as described in the previous attack. Thus, about $2^{109} \cdot 2^{31} \cdot 2^{-16} \cdot 8 = 2^{127}$ pairs are selected.

The following calculations are similar to those described in the previous subsection. Every selected pair suggests about 2^{16} possible values for $KW1 \oplus KA$ and about 64 possible values of 16 bits (bytes '13' and '33') of $KW2 \oplus KB$. In total, about $2^{16} \cdot 64 \cdot 2^{127} = 2^{149}$ values are suggested. The 200 right pairs must suggest the right values of $KW1 \oplus KA$ and of 16 bits of $KW2 \oplus KB$. So we look for values that are suggested 200 or more times. We expect that approximately $\binom{2^{149}}{200} \cdot (2^{-144})^{199} \cong 2^{-101}$ wrong values are suggested by 200 or more pairs. We observe that the subkey KA has only 2^{32} possible values (see Appendix A). For each guessed KA, we find $KW1$ by XOR-ing this KA with the received combination. According to the key scheduling procedure described

in Appendix A, we can recover all the subkeys and the key given any two consecutive 128-bit subkeys in time of a single encryption. $KW1$ and KAB are two consecutive subkeys. We receive $KW1$ and 32 bits from KAB (KA is word 0 of KAB taken four times), so it remains to guess additional 96 bits of KAB to recover all the subkeys and the original key. The wrong keys can be discarded by trial decryption.

The attack requires 2^{125} chosen ciphertexts, and the complexity of analysis is about $2^{32} \cdot 2^{96} = 2^{128}$.

5.3 Improved Attack for 192-Bit Keys

We can improve the complexity of the previous attack for 192-bit keys. This improvement uses the same structures and pair selection process as in the previous subsection, and changes only the key recovery process.

From the previous section, we expect to get about 1 value of $KW1 \oplus KA$ suggested by 200 or more pairs. We have 2^{32} possible combinations of subkeys ($KW1$,KA) as described in the previous subsection. According to the key scheduling algorithm, the knowledge of $KW1$ and KH is sufficient to recover all the other subkeys and the original key. Due to the fact that KH has only 64 non-zero bits for 192-bit keys, we can recover all the subkeys and the original key in $2^{32} \cdot 2^{64} = 2^{96}$ steps. Wrong keys can be discarded by comparing the guessed KA with KA received from $KW1$ and KH. If more than one key remains after this comparison then we discard all the remaining wrong keys by trial decryption.

The improved attack requires 2^{125} chosen ciphertexts, and the complexity of analysis is about 2^{96}.

6 Conclusions

In this paper we demonstrated that both the 8-round version and the 9-round version of Q are vulnerable to differential cryptanalysis. While these attacks are academic in nature, they show that the security analysis of the designer was insufficient. The most important improvements are the usage of differentials instead of characteristics, and the usage of characteristics that are not iterative, but 'almost iterative'.

The analysis shows that the combination of elements from two secure ciphers, in casu Rijndael and Serpent, does not necessarily result in a secure cipher. The round function of Q was composed by combining the nonlinear transformations of Rijndael and Serpent, and leaving out all the linear transformations except for the permutation. As a result, the diffusion properties of the round transformation of Q are suboptimal. Whereas the designers of Rijndael and Serpent ensured that there would exist no multi-round characteristics with a small number of active S-boxes in the original ciphers, several such characteristics can be defined for Q. Furthermore, the limited diffusion in the round transformation of Q allows combination of many characteristics into one differential.

References

[1] Ross Anderson, Eli Biham, Lars Knudsen, *Serpent: A proposal for Advanced Encryption Standard*, submitted to AES, 1998.
[2] Eli Biham, Adi Shamir, *Differential cryptanalysis of the Data Encryption Standard*, Springer Verlag, 1993.
[3] Joan Daemen, Vincent Rijmen, *The block cipher Rijndael*, Smart Card Research and Applications, LNCS 1820, J.-J. Quisquater and B. Schneier, Eds., Springer-Verlag, 2000, pp. 288-296.
[4] Xuejia Lai, James L. Massey,*Markov Ciphers and Differential Cryptanalysis*, proceedings of EUROCRYPT'91, LNCS 547, pp.17-38, 1991.
[5] Leslie 'Mack' McBride, *Q: A Proposal for NESSIE v2.00*, submitted to NESSIE, 2000.
[6] www.cryptonessie.org

A The Key Scheduling of Q

Generally, the cipher Q may work with a key of any length. Keys longer than 256 bits are reduced to 256 bits using the polynomial $X^{256} + X^{193} + X^{113} + X^6 + 1$ in GF(2^{256}). The other keys are expanded to the 256-bit keys by adding zeroes as the most significant bits. The 128 least significant bits of the received key material are called KL, and the 128 most significant bits are called KH. Thus for 128-bit keys $KH = 0$ and for 192-bit keys the 64 most significant bits of KH are zeroes.

Then the procedure described later runs $r + 4$ times, where r is a number of rounds in Q, and their 128-bit outputs are taken as subkeys in the following order:

$$Discard, KW1, KAB, K_0, K_1, \ldots, K_{r-1}, KW2.$$

The first output is discarded and the subkeys KA, KB are built from the words 0 and 1 of KAB respectively. In both, the corresponding 32-bit word is used four times with the permutation applied.

The procedure description:

1. A single byte counter is XOR-ed to the least significant byte of KL (a counter is started from 0, and is increased for every procedure running).
2. $KL = KL \oplus KH$.
3. Byte Substitution layer is performed on KL.
4. The constant $0x9e3779b9$ is XOR-ed to the word 0 of KL.
5. Bit-Slice layer (S-box C) is performed on KL.
6. Permutation layer is performed on KL.
7. Output KL.

Note that for 128-bit keys, the knowledge of any 128-bit subkey suffices to recover all other subkeys and the original key. For 192-bit keys and 256-bit keys the knowledge of any two consecutive 128-bit subkeys suffices to recover all other subkeys and the original key.

Differential Cryptanalysis of Nimbus*

Vladimir Furman

Computer Science Department, Technion - Israel Institute of Technology, Haifa 32000, Israel. vfurman@cs.technion.ac.il.

Abstract. Nimbus is a block cipher submitted as a candidate to the NESSIE project by Alexis Machado. Like many other ciphers Nimbus combines multiplication operations with XOR operations, a common technique to protect against various kinds of cryptanalysis. In this paper we present two new differential properties of multiplication operations with probability about 1/2 which we use to design a one-round iterative characteristic of Nimbus. We iterate it to a characteristic of the full cipher with probability 1/32, which in turn we use to attack the full cipher and find all the key material using 256 chosen plaintexts and 2^{10} complexity. Thus, we show that the inclusion of multiplication operations in a cipher does not necessarily protect against attacks.

1 Introduction

Nimbus [2] is a block cipher submitted as a candidate to the NESSIE project by Alexis Machado. The cipher has 64-bit blocks and 128-bit keys. It consists of five rounds of the form

$$Y_i = K_i^{odd} \cdot g(Y_{i-1} \oplus K_i),$$

where i is the round number, K_i and K_i^{odd} are subkeys (K_i^{odd} is always odd), '$\oplus$' is exclusive-OR (XOR), g is the bit-reversal function, '$\cdot$' is multiplication modulo 2^{64}, Y_0 is the plaintext, and Y_5 is the ciphertext.

The multiplication operation used in Nimbus was considered to be a "good" operation against differential cryptanalysis. The main "bad" property that was known about multiplication operation was that it preserves the least significant bits of form $1\underbrace{0\ldots0}_{i}$ for any i. Therefore, the cipher with the multiplication operation together with the bit-reversal function were considered to have a "good" defense against cryptanalysis.

In this paper we present two new differential properties of the multiplication operation with probability slightly larger than 1/2. In addition, the input/output difference in one of them has a palindromic form. Hence this differential property is not affected by the bit-reversal function. We also describe a set of new

* The work described in this paper has been supported by the European Commission through the IST Programme under Contract IST-1999-12324 and by the fund for the promotion of research at the Technion.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 187–195, 2002.

differential properties of the multiplication operation with probabilities that are bigger than random probability.

We use these differential properties to describe a 1-round iterative differential characteristic [1] with probability 1/2, whose iteration to the full cipher has a probability of 1/32. We use this characteristic to break Nimbus using 256 chosen plaintexts and 2^{10} complexity.

The paper is organized as follows: In Section 2 we describe the new differential properties of multiplication. In Section 3 we use them to devise a 1-round iterative characteristic of Nimbus. In Section 4 we present a chosen plaintext attack on the full Nimbus, and in Section 5 we show a chosen ciphertext attack, which is similar to the chosen plaintext attack. Finally, in Appendix A we describe a subkey recovering procedure that is used in the attacks.

2 New Multiplication Characteristics

Let A, B and K^{odd} be n-bit random numbers, where K^{odd} is odd.

Lemma 1. *[given without a proof] Let A and B be n-bit integers.*

- *If the least significant bits of A and B are 0 then*

$$A \oplus B = 0\underbrace{1\ldots1}_{n-2}0 \Leftrightarrow A + B = 0\underbrace{1\ldots1}_{n-2}0,$$

  where the numbers on the right hand sides of the equalities are given in a binary notation.
- *If the least significant bits of A and B are 1 then*

$$A \oplus B = 0\underbrace{1\ldots1}_{n-2}0 \Leftrightarrow A + B = 1\underbrace{0\ldots0}_{n-1}.$$

Claim 1: Let A, B and K^{odd} be n-bit random integers such that K^{odd} is odd. Then

$$A \oplus B = 0\underbrace{1\ldots1}_{n-2}0 \Rightarrow (A \cdot K^{odd}) \oplus (B \cdot K^{odd}) = 0\underbrace{1\ldots1}_{n-2}0 \tag{1}$$

holds with probability $1/2 + 2/2^n$, and also

$$A \oplus B = \underbrace{1\ldots1}_{n-1}0 \Rightarrow (A \cdot K^{odd}) \oplus (B \cdot K^{odd}) = \underbrace{1\ldots1}_{n-1}0 \tag{2}$$

holds with the same probability.

Proof: Without loss of generality, it is sufficient to prove the first equation; the proof for the second equation is similar.

We observe that multiplication by an odd number preserves the least significant bit. Assume $A \oplus B = 0\underbrace{1\ldots1}_{n-2}0$. There are two cases:

1. The least significant bits of A and B are 0.
   By Lemma 1, the sum $A + B$ is $0\underbrace{1\ldots1}_{n-2}0$ (i.e., $2^{n-1} - 2$). Hence, the sum $A \cdot K^{odd} + B \cdot K^{odd} = (A + B) \cdot K^{odd}$ is $(2^{n-1} - 2) \cdot K^{odd}$ for any odd K^{odd}. On the other hand, when the right hand side of Equation (1) holds, the same lemma ensures that $(A+B) \cdot K^{odd} = 2^{n-1} - 2$. Therefore, $(2^{n-1} - 2) \cdot K^{odd} = 2^{n-1} - 2$. This is satisfied when K^{odd} is either 1 or $2^{n-1} + 1$. As K^{odd} can obtain 2^{n-1} odd values, it happens with a probability of $2/2^{n-1}$.
2. The least significant bits of A and B are 1.
   By Lemma 1, the sum $A + B$ is $1\underbrace{0\ldots0}_{n-1}$ (i.e., 2^{n-1}). Hence, the sum $A \cdot K^{odd} + B \cdot K^{odd} = (A + B) \cdot K^{odd}$ is 2^{n-1} for any odd K^{odd}. By the same lemma, $(A \cdot K^{odd}) \oplus (B \cdot K^{odd})$ is always $0\underbrace{1\ldots1}_{n-2}0$. Therefore, it happens with probability 1.

Each case happens with a probability of 1/2, so the total probability is $1/2 + 2/2^n$. □

We conclude that Equation (1) and Equation (2) are always (with a probability of 1) satisfied for $K^{odd} = 1$ and $K^{odd} = 2^{n-1} + 1$. For all the other values of K^{odd} they are satisfied with probability 1/2.

Now we give a more general claim, that may be proved in a similar way.

Claim 2: Let A, B and K^{odd} be n-bit random variables such that K^{odd} is odd, and let $i, j \geq 0$, $i + j \leq n - 2$. Then

$$\begin{aligned} A \oplus B = \underbrace{0\ldots0}_{i}\ \underbrace{1\ldots1}_{n-2-i-j}\ 10\ \ \underbrace{0\ldots0}_{j} \Rightarrow \\ (A \cdot K^{odd}) \oplus (B \cdot K^{odd}) = \underbrace{0\ldots0}_{i}\ \underbrace{1\ldots1}_{n-2-i-j}\ 10\ \ \underbrace{0\ldots0}_{j} \end{aligned} \tag{3}$$

holds with probability

$$\begin{cases} 1/2^{j+1} \cdot 1/2^{i-1} + 1/2^{n-2-j} \cdot (1 - 2^{-(j+1)}), & for\ i \geq 1, \\ 1/2^{j+1} + 1/2^{n-2-j} \cdot (1 - 2^{-(j+1)}), & for\ i = 0. \end{cases}$$

In the case where the XOR difference has a palindromic form we obtain the following claim:

Claim 3: Let A, B and K^{odd} be n-bit random variables, and $0 \leq 2 \cdot i \leq n - 4$. Then

$$\begin{aligned} A \oplus B = \underbrace{0\ldots0}_{i}\ 01\ \underbrace{1\ldots1}_{n-4-2\cdot i}\ 10\ \ \underbrace{0\ldots0}_{i} \Rightarrow \\ (A \cdot K^{odd}) \oplus (B \cdot K^{odd}) = \underbrace{0\ldots0}_{i}\ 01\ \underbrace{1\ldots1}_{n-4-2\cdot i}\ 10\ \ \underbrace{0\ldots0}_{i} \end{aligned} \tag{4}$$

holds with probability $1/2^{2\cdot i+1} + 1/2^{n-2-i} \cdot (1 - 2^{-(i+1)})$.

3 A One-Round Iterative Characteristic

Nimbus has 64-bit words, so the characteristics of the multiplication operation described in Claim 2 have a probability of $1/2 + 2/2^{64}$. Moreover, the characteristics' probability are key dependent: when the subkeys 1 and $2^{63}+1$ are used in the multiplication operation the characteristics are always satisfied (probability 1). When other subkeys are used in the multiplication operation the probability of the characteristics is exactly 1/2.

The characteristic from Equation (1) has a palindromic form, so the g function always preserves the difference $0\underbrace{1\ldots1}_{n-2}0$. Hence, we get a one-round iterative characteristic $0\underbrace{1\ldots1}_{n-2}0 \rightarrow 0\underbrace{1\ldots1}_{n-2}0$ with a probability of $1/2 + 2/2^{64}$.

Nimbus has five rounds, so the characteristic $0\underbrace{1\ldots1}_{n-2}0 \rightarrow 0\underbrace{1\ldots1}_{n-2}0$ of the full cipher has a probability of $(1/2 + 2/2^{64})^5 \cong 2^{-5}$. For most keys the probability of the characteristic is 2^{-5}. There are, however, a few keys for which the characteristic has probabilities 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, and 1.

4 A Chosen Plaintext Attack

Denote $\Delta = 0\underbrace{1\ldots1}_{n-2}0$. We generate structures of 64 plaintexts as follows: We randomly choose 32 plaintexts whose least and most significant bits are 0. We XOR each of these plaintexts with Δ, and get 32 pairs of plaintexts. Then, we build three additional structures by:

1. Complementing the most significant bits of all the plaintexts.
2. Complementing the least significant bits of all the plaintexts.
3. Complementing the most and the least significant bits of all the plaintexts.

We get four structures of 32 pairs of plaintexts each, and request to encrypt these 256 plaintexts to their ciphertexts under the unknown key.

Exactly two structures lead to difference Δ after one round. These two structures differ only by complementing the least significant bits of their plaintexts. Exactly one structure from these two structures leads to difference Δ after two rounds. We call such structure *Δ-structure*.

The iterative characteristic described in the previous section predicts that the pairs of plaintexts that belongs to the Δ-structure, have difference Δ after five rounds with a probability of 1/8. Thus, we have about $32 \cdot 1/8 = 4$ pairs from this structure that cause a difference of Δ after five rounds according to the above characteristic. Randomly, a pair (from any structure) can lead to a difference of Δ after five rounds with a probability of 2^{-64}, so we do not expect any wrong pair to lead to difference Δ after five rounds. Based on the ciphertext differences we can easily determine the which structure is the Δ-structure. We can, therefore, conclude which structure has difference Δ after one round, but

not after two rounds, and which structures do not have a difference of Δ after one round.

We continue the analysis with the 32 pairs of the Δ-structure. In this step of the attack we use the two-round characteristic $\Delta \to \Delta$. We use this characteristic starting from the third round. Given the ciphertexts of the above structures we find the subkey K_5^{odd} as follows:

1. The two-round characteristic $\Delta \to \Delta$ has probability 1/4, so a pair with difference Δ before the third round has difference Δ after round 4 with probability 1/4, i.e., there are 8 pairs on average with difference Δ after round 4. Each such pair does not lead to difference Δ after the fifth round with probability 1/2. Thus, we have 4 pairs on average with difference Δ after four rounds but without difference Δ after the fifth round. We call such a pair a *matching pair*.
2. We can recognize the matching pairs by testing that the following condition is satisfied, and discarding all the pairs that fail this test.

   Condition 1: All the three criteria are satisfied for matching pairs:
   - Multiplication by an odd number preserves the least significant bits of the form $1\underbrace{0\ldots0}_{i}$, where $i \geq 0$. Thus, matching pairs must have the bits 10 as the two least significant bits of the ciphertext XOR difference.
   - The matching pairs must have 0 as the least significant bit of the ciphertexts (otherwise, it would lead to difference Δ after five rounds as described in Section 2).
   - All matching pairs must have the same third least significant bit of their ciphertext XOR difference, because (for matching pairs) this bit depends only on the second least significant bit of the subkey used in multiplication.
     Note that we use this criterion when, there is a majority of matching pairs. Hence, we can recognize the right value of this bit, and in wrong pairs the corresponding bit equals to the right value with a probability of 1/2.

   This condition is satisfied randomly with a probability of $1/4 \cdot 1/2 \cdot 1/2 = 1/16$, hence we have about $(32 - 8) \cdot 1/16 \cong 2$ wrong pairs that satisfy Condition 1.
3. We select the pairs that satisfy Condition 1 (matching and wrong pairs). For each selected pair we solve the equation

$$0\underbrace{1\ldots1}_{n-2}0 \cdot K' = C_1 + C_2,$$

   where C_1, C_2 are the ciphertexts of this pair. We have 4 matching pairs that must suggest the same K' and only two wrong pairs that may suggest other values. Hence, we expect that K_5^{odd} is the most frequently suggested K'. Note that we do not know the most significant bit of K_5^{odd} from the above equation. So we actually have two possible subkeys K_5^{odd} which differ by the

most significant bit. It suffices to work only with one of them, and choose the right one in the later stage of the attack.

Given K_5^{odd} we find the subkeys K_5 and K_4^{odd} using the one-round characteristic $\Delta \to \Delta$. We use the same structure as in the previous stage, and decrypt the received ciphertexts by a half round using the recovered K_5^{odd}. We call these values *partially decrypted values*. Each pair leads to difference Δ after the third round, but does not lead to difference Δ after the 4th round with a probability of $1/2 - 1/4 = 1/4$. In this stage of the attack such a pair is called a *matching pair*. There are about $32 \cdot 1/4 = 8$ matching pairs. According to the partially decrypted values of the pairs that lead to difference Δ after 4 rounds, we can find the least significant bit of K_5. Using this knowledge we select the pairs according to Condition 1, except that here we are not looking at ciphertexts but at partially decrypted values. Wrong pairs may satisfy Condition 1 with probability 1/16. So only about $(32 - 16) \cdot 1/16 = 1$ wrong pair is selected. For the selected pairs (matching and wrong), we have the following equation:

$$\begin{cases} (A \cdot K') \oplus K'' = C \\ (B \cdot K') \oplus K'' = D \end{cases} \tag{5}$$

where A, B, K' and K'' are unknown values with $A \oplus B = \Delta$, and C, D are the partially decrypted values. We use the procedure described in Appendix A to find K' and K''. This procedure takes 4 pairs, solves the system of the above equations for these 4 pairs, and returns a set of values (K',K''). There are about 9 selected pairs (8 matching and 1 wrong). We randomly take 4 pairs (from the 9), run the procedure described in Appendix A and obtain a set of values. Then we take another subgroup of 4 pairs (from the 9), and run the procedure described in Appendix A on these pairs. If the returned set has an intersection with a previous set, we take the intersection as a new set instead of these two sets. We continue the process with other subgroups until the intersection of some sets gives the set of combinations which differ only by bits that we cannot uniquely identify by additional running of the procedure (see detailed description in Appendix A). The received set of candidates is expected to contain the correct value of (K_4^{odd},K_5), and it consists of 512 combinations on average. The probability that the received set of candidates does not contain the correct value is negligible. On average, about 6 subsets should be analyzed to identify the set that contains the correct value of (K_4^{odd},K_5). We cannot identify the correct value uniquely at this stage. We discard the wrong values from this set later in the attack.

All the other subkeys of rounds $1, \ldots, 4$ can be found in a similar way. We now show briefly how we do it efficiently.

For finding K_3^{odd} and K_4, we use the same structure as before. In this stage of the attack, the pairs causing difference Δ after two rounds and not after three rounds are called *matching pairs*. About half of the pairs are matching pairs, so we have $32 \cdot 1/2 = 16$ matching pairs. All the pairs from this structure have difference Δ after two rounds, and those of them that have difference Δ after three rounds were used in the previous stages of the attack. Hence, all the

remaining pairs (which do not have difference Δ after round 3) are the matching pairs. At this point we have $2 \cdot 512 = 1024$ possible combinations of the subkeys (K_4^{odd},K_5,K_5^{odd}). The right subkey must decrypt each matching pair to values with an XOR difference that has two least significant bits 10 (the first criterion of Condition 1). A wrong subkey combination passes this criterion with probability $(2^{-2})^{16} = 2^{-32}$ (for all 16 matching pairs). Thus, we can discard all wrong subkey combinations (K_4^{odd},K_5,K_5^{odd}), except for one wrong subkey combination that differs from the right subkey combination by the most significant bit of K_4^{odd}. This bit does not influence the analysis in this stage of the attack, so it is sufficient to continue the analysis only with one of the combinations, and identify the right combination in a later stage of the attack. Given the combination we choose to analyze, we decrypt the partially decrypted values by one more round. From now on, these values are called partially decrypted values. We run the procedure described in Appendix A on 4 randomly chosen pairs among the 16 matching pairs and intersect the resulting sets as in the previous case. About three runs of this procedure are needed to obtain a set of values that are expected to contain the correct value of (K_3^{odd},K_4). The wrong values from the remaining set are discarded later in the attack.

For finding K_2^{odd} and K_3, we use the structure that leads to difference Δ after one round but not to difference Δ after two rounds. As in the previous stage, we discard almost all wrong subkey combinations and have only two possibilities for (K_3^{odd},K_4,K_4^{odd},K_5,K_5^{odd}) which differ by the most significant bit of K_3^{odd}. This bit does not influence the analysis in this stage of the attack, so it is sufficient to continue the analysis only with one of the combinations, and identify the right combination in a later stage of the attack. Given the combination we choose to analyze, we decrypt the partially decrypted values by one more round, and run Appendix A on four pairs randomly chosen among the 32 pairs from this structure. About three runs of this procedure are needed to obtain the set of values that are expected to contain the correct value of (K_2^{odd},K_3). The wrong values from this set are discarded later in the attack.

Finally, we have the equation:

$$\begin{cases} ((A \oplus K) \cdot K') \oplus K'' = C \\ ((B \oplus K) \cdot K') \oplus K'' = D. \end{cases} \tag{6}$$

For finding the remaining subkeys (K_1,K_1^{odd},K_2), we use the pairs from the two structures that do not lead to difference Δ after one round. As in the previous stage, we discard almost all wrong subkey combinations and have only two possibilities of (K_2^{odd},K_3,K_3^{odd},K_4,K_4^{odd},K_5,K_5^{odd}) which differ by the most significant bit of K_2^{odd}. This bit does not influence the analysis in this stage of the attack, so it is sufficient to continue the analysis only with one of the combinations, and identify the right combination in a later stage of the attack. Given the combination we choose to analyze, we decrypt the partially decrypted values by one more round. Equation (6) is similar to Equation (5), so we find K' and K'' by running Appendix A on four pairs randomly chosen among the 64 pairs from these structures. About three runs of this procedure are needed to obtain the set

of values that are expected to contain the correct value of (K_1^{odd},K_2). For each received combination we can easily find an expected value of K_1. Thus we obtain a set of values that are expected to contain the correct value of (K_1,K_1^{odd},K_2).

We finally have $2 \cdot 512 = 1024$ possible combinations of all subkeys. The right subkeys may be found by encrypting one or two plaintexts.

In this attack we completely recover all the subkeys of Nimbus, which suffice to encrypt and decrypt without knowing the original key. The attack requires 256 chosen plaintexts with a complexity equivalent to 2^{10} full cipher encryptions.

5 A Chosen Ciphertext Attack

The chosen ciphertext attack works in a similar way, except that we know exactly which structures lead to difference Δ after one round, because in decryption the multiplication is the first operation. We need only two structures (in which the least significant bit is 1), i.e., only 128 ciphertexts, to find all the subkeys, except for K_5, K_5^{odd}. According to Appendix A, we need about 4 additional pairs (8 ciphertexts) in order to find these subkeys.

This attack requires 136 chosen ciphertexts, and its complexity is at most 2^{10} full cipher encryptions.

References

[1] Eli Biham, Adi Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer Verlag, 1993.
[2] Alexis Warner Machado, *The Nimbus Cipher: A Proposal for NESSIE*, NESSIE Proposal, September 2000.

A Recovering K' and K''

Given four pairs, we recover a set of options for K' and K'' by solving the system of eight equations received from combination of the Equation (5) of the four pairs. For pair $i \in 0 \ldots 3$, the equations are:

$$\begin{cases} (A_i \cdot K') \oplus K'' = C_i \\ (B_i \cdot K') \oplus K'' = D_i \end{cases}$$

where A_i, B_i, K' and K'' are unknown, and C_i, D_i are known values. In addition, we know that for all the pairs $A_i \oplus B_i = \Delta$, K' is an odd number, and the least significant bits of A_i and B_i are 0.

K' is an odd number, so its least significant bit is 1, and thus the multiplication of some number by K' preserves the least significant bit of this number. The least significant bit of A_0 and B_0 is 0, so we can easily find the least significant bit of K'' from C_0 and D_0. In contrast, we may get both values for the second least significant bit of K'', and we cannot identify this bit by this procedure. We can find the second least significant bit of K' in a deterministic

way: $1 \oplus$ the third least significant bit of $(C_0 \oplus D_0)$. If this bit is zero, then we can find the next (third) least significant bit of K' in a similar way using the 4th least significant bits of C_0 and D_0. If the third least significant bit is zero as well, we can find the next (fourth) least significant bit of K' in a similar way, and so on till the first 1 appears. We denote the number of such found zero bits of K' by Z. Note that for Z least significant bits of K'', starting from the third least significant bit, we cannot identify the correct value using this procedure.

When we meet the first 1 we run the following procedure: We need at least four pairs for the following operations. We start by finding the first two bits of K' that cannot be found in a previous step. We find these bits in both numbers (K' and K'') together, as follows:

- We guess the next 3 least significant bits of A_0 for one pair and the next 2 least significant bits of K'. Note that the most significant bit, among the guessed bits of A_0, is used only for better identifying the remaining bits, and is not remembered in the next step.
- From A_0 we get the 3 corresponding bits of B_0 (as we know $A_0 \oplus B_0 = \Delta$), multiply both numbers by the guessed K' (i.e., all the bits of K' guessed so far) and check if the XOR difference of results are equal to the XOR difference of C_0 and D_0. If it is not equal, then we continue with the next guess. Otherwise, we find the bits of K'' ($K'' = C_0 \oplus (A_0 \cdot K')$), decrypt the other pairs using the received K' and K'', and check that the received A_i and B_i have the required XOR difference.

There are at most 31 such steps. In this procedure, the following bits may obtain any possible values:

- complementing any of the $Z + 1$ least significant bits of K'' (starting from the second least significant bit),
- complementing the most significant bit of K',
- replacing K' by $K' \oplus (\underbrace{1 \ldots 1}_{61-Z} \ll (Z + 2))$,
- complementing the most significant bit of K''.

Hence we get 2^{Z+4} possible combinations of (K',K''), or about 512 possible combinations on average. If the four given pairs were chosen badly, then we can obtain some "noise" - additional unidentified bits of (K',K''). We can discard the wrong values of these bits by running this procedure with other pairs. We need to run the procedure about 3 times on average to discard the "noise".

All combinations of (K',K'') returned by this procedure have strong relations with themselves. If we have one of the possible combinations of (K',K'') we can obtain the others by complementing the corresponding bits. So it is sufficient to find one of them and to calculate all the others.

The whole procedure takes up to $31(steps) \cdot (2^3 \cdot 2^2)(guesses) \cdot 8$ (4 pairs) $\cong 2^{13}$ operations which are equivalent to about 2^8 one-round computations.

Fast Correlation Attack Algorithm with List Decoding and an Application

Miodrag J. Mihaljević[1,2], Marc P.C. Fossorier[3], and Hideki Imai[4]

[1] Mathematical Institute, Serbian Academy of Sciences and Arts,
Kneza Mihaila 35, 11001 Belgrade, Yugoslavia
miodragm@turing.mi.sanu.ac.yu
[2] SONY Computer Science Laboratories
Takanawa Muse Bld., 3-14-13, Higashi-Gotanda, Shinagawa-ku,
Tokyo, 141-0022 Japan
[3] Department of Electrical Engineering, University of Hawaii,
2540 Dole St., Holmes Hall 483, Honolulu, HI 96822, USA
marc@spectra.eng.hawaii.edu
[4] University of Tokyo, Institute of Industrial Science,
4-6-1, Komaba, Meguro-ku, Tokyo, 153-8505 Japan
imai@iis.u-tokyo.ac.jp

Abstract. An improved method for the fast correlation attack on certain stream ciphers is presented. The proposed algorithm employs the following decoding approaches: list decoding in which a candidate is assigned to the list based on the most reliable information sets, and minimum distance decoding based on Hamming distance. Performance and complexity of the proposed algorithm are considered. A desirable characteristic of the proposed algorithm is its theoretical analyzibility, so that its performance can also be estimated in cases where corresponding experiments are not feasible due to the current technological limitations. The algorithm is compared with relevant recently reported algorithms, and its advantages are pointed out. Finally, the proposed algorithm is considered in a security evaluation context of a proposal (NESSIE) for stream ciphers.

Keywords. Stream ciphers, keystream generators, linear feedback shift registers, nonlinear combiner, nonlinear filter, cryptanalysis.

1 Introduction

An important method for attack or security examination of certain stream ciphers based on the nonlinear combination keystream generators and the nonlinear filter keystream generators (see [13], for example) is the fast correlation attack proposed in [14] and [22] as a significant improvement of the basic correlation attack [20]. Further extensions and refinements of fast correlation attack, as well as its analysis are presented in a number of papers including the following most recent ones: [8], [9], [7], [3], [15], [2], [16], [17]-[18], and [10]. The basic ideas

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 196–210, 2002.

of the reported fast correlation attacks explicitly or implicitly include the following two main steps: (i) transform the cryptographic problem into a suitable decoding one; and (ii) apply (devise) an appropriate decoding algorithm. The decoding step employs either one-step or iterative decoding techniques.

Recently two techniques for fast correlation attack based on one-step decoding have been proposed in [3] and [16] demonstrating low-complexity and high-performance, and very recently another one-step based method for the fast correlation attack through reconstruction of linear polynomials has been reported in [10], showing powerful performance and low complexity. Also, a high-performance iterative decoding technique for fast correlation attack have been reported in [2] and [17]. All these approaches do not depend on the LFSR feedback polynomial weight.

Having in mind these recent achievements, the main objective of this paper is to propose an improved noniterative decoding approach for the fast correlation attack and to compare it with the recently reported results. The proposed approach employs list and minimum distance decoding, and it is based on the most reliable information sets.

The paper is organized as follows. Section 2 summarizes the decoding approach for fast correlation attack. Section 3 points out the main underlying ideas for the construction of an improved algorithm for the cryptanalysis. The complete algorithm for the cryptanalysis is proposed in Section 4. The main characteristics of the algorithm are discussed in Sections 5 and 6. An application of the algorithm for the security evaluation of a proposal for stream ciphers (NESSIE: LILI-128) is given in Section 7. Finally, the results of this paper are summarized in Section 8.

2 Background

2.1 Decoding Model of the Cryptanalysis

A nonlinear combination keystream generator combines the outputs of several linear feedback shift registers (LFSRs) by a nonlinear Boolean function. A nonlinear filter keystream generator consists of a single LFSR and a nonlinear Boolean function whose inputs are taken from some LFSR stages to produce the output. Both schemes may become vulnerable due to the correlation between an LFSR and the generator output.

The term correlation means that the mod 2 sum of the corresponding outputs of an LFSR and a 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 (a similar consideration is valid for the nonlinear filter, as well):

- 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.

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, $z_n = x_n \oplus e_n$, $n = 1, 2, ..., N$, where the effect of the BSC with crossover 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.

2.2 List Decoding

List decoding was introduced in [4] and [21] and random coding arguments were used to explore its average decoding error probability for block codes of low rates for the BSC (for more details see [5] and [6], for example).

In list decoding of a block code, the decoder gives as its output not one codeword but a list of possible candidate codewords. The list decoder is in error only if the correct codeword is not on the list.

When the list is composed of ℓ candidates the decoding is denoted as list-of-ℓ decoding.

3 Underlying Ideas for Cryptanalysis

The developed algorithm is based on a pre-processing stage and a processing stage performed as follows.

3.1 Pre-processing Stage

The central issue in the pre-processing stage is the construction of parity-check equations which follows the approach proposed in [15].

An LFSR can be considered as a linear finite state machine, and accordingly, a state of a length-L LFSR after t clocks is given by the following matrix-vector product over GF(2):

$$\mathbf{x}_t = \mathbf{A}^t \mathbf{x}_0 \ , \ t = 1, 2, ... \ , \tag{1}$$

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, and $\mathbf{A}^t$ is the t-th power over GF(2) of the $L \times L$ state transition 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:

$$\mathbf{A} = \begin{bmatrix} b_1 & b_2 & b_3 & ... & b_L \\ 1 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & ... & . \\ . & . & . & ... & . \\ 0 & & & ... 1 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \mathbf{A}_3 \\ . \\ \mathbf{A}_L \end{bmatrix} ,$$

where each $\mathbf{A}_i$, $i = 1, 2, ..., L$, represents a $1 \times L$ binary matrix (a row-vector).

The 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. Consequently, assuming that the first B information bits are known due to employment of a partial exhaustive search, for the i-th codeword bit, $i = B+1, B+2, ..., D$, where D is number of codeword bits under consideration, the novel algorithm employs the parity-check equations constructed according to the following definition.

Definition 1. The set Ω_i of parity-check equations associated with bit-i is composed of:

- For $i = B + 1, B + 2, ..., L$:
  All parity-check equations obtained as the *mod*2 sum of two distinct 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.
- For $i = L + 1, L + 2, ..., D$:
  All parity-check equations obtained as the *mod*2 sum of three distinct basic parity-check equations,

  $$(z_i \oplus \mathbf{A}_1^i \mathbf{z}_0) \oplus (z_m \oplus \mathbf{A}_1^m \mathbf{z}_0) \oplus (z_n \oplus \mathbf{A}_1^n \mathbf{z}_0) ,$$

  where m and n have arbitrary values providing that the vector sum $\mathbf{A}_1^i \oplus \mathbf{A}_1^m \oplus \mathbf{A}_1^n$ has arbitrary values in the first B coordinates, and value zero in the all other coordinates.

Note that the previous definition differs from these given in [15], and also that for given parameters L, B and D, the sets Ω_i, $i = B + 1, B + 2, ..., D$, can be constructed in advance through a search procedure in a preprocessing phase, and later used for any particular application with these given parameters.

According to [15] and [17], in any set Ω_i specified by Definition 1, $i = B + 1, B + 2, ..., D$, an approximation on the expected number $\mid \bar{\Omega} \mid$ of parity-checks in Ω_i is given by:

$$\mid \bar{\Omega} \mid \approx 2^{B-L}\binom{N-L}{2} \text{ or } 2^{B-L}\binom{N-D}{2} \approx 2^{B-L}\binom{N}{2}. \quad (2)$$

3.2 Processing Stage

The processing stage employs list decoding to form a list of candidates and the minimum distance decoding (MDD) to select the true candidate. The list of candidates is formed based on a directed search and most reliable information sets. Accordingly, the processing consists of the following steps:
– Setting the hypothesis;
– Parity-checks evaluation;
– Selection of the most reliable information sets and assigning the list decoding candidate;
– MDD: correlation check.

4 Algorithm for Fast Correlation Attack Based on List and Minimum Distance Decoding

INPUT:
– the generator output $\{z_i\}_i^N$;
– the LFSR feedback polynomial;
– the correlation noise probability p, the missing event probability P_m and the false alarm probability P_f;
– the algorithm parameters: B and D.

PRE-PROCESSING

- *Setting of the algorithm parameters M and T*:
  for given p, P_m and P_f determine (according to [20]) the required number M of bits for MDD (i.e. the correlation check), and the MDD threshold T.
- *Determination of the parity-check equations*:
  for each codeword bit i, $i = B + 1, B + 2, ..., D$, construct the set Ω_i of corresponding parity-check equations based on Definition 1.

PROCESSING: List-of-2^{B+1} decoding and MDD employing the most reliable information sets.

1. *Setting the hypothesis*
   - Set a new hypothesis on the first B coordinates of the LFSR initial state, i.e. previously not considered pattern $\hat{x}_1, \hat{x}_2, ..., \hat{x}_B$, for the first B information bits, based on z_i, $i = 1, 2, ..., B$ and a new, previously unconsidered, most likely error-pattern.
   - If no one new pattern is possible go to Output (b).

2. *Parity-checks evaluation*
   For each codeword bit position i, $i = B+1, B+2, ..., D$, calculate the parity-check values employing the parity check equations in the set Ω_i.
3. *Selection of the most reliable information sets*
   Based on the evaluation of the parity-check equations on the codeword bit position i, $i = B+1, B+2, ..., D$, form two most reliable estimations of all information bits according to the following.
   - select $L-B$ positions corresponding to the bits with the most **satisfied** parity checks, and assuming that all of them are correct recalculate the information bits on coordinates $i = B+1, B+2, ..., L$.
   - select $L-B$ positions corresponding to the bits with the most **unsatisfied** parity checks, and assuming that all of them are in error, complement them and recalculate the information bits on coordinates $i = B+1, B+2, ..., L$.
4. *MDD - Correlation check*
   For each of two estimations obtained from Step 3, check if the current estimation of the information bits $\hat{\mathbf{x}}_0 = [\hat{x}_1, \hat{x}_2, ..., \hat{x}_L]$, is the true one, according to the following:
   For a given $\hat{\mathbf{x}}_0$, generate the corresponding sequence $\hat{x}_1, \hat{x}_2, ..., \hat{x}_M$, and calculate
   $S = \sum_{n=1}^{M} \hat{x}_n \oplus z_n$.
   If $S \leq T$ go to the Output (a), otherwise go to Step 1.

OUTPUT:
(a) The considered vector $\hat{\mathbf{x}}_0$ of information bits accepted as the true one;
(b) The true vector of information bits is not found.

5 Performance and Complexity Analyses

5.1 Performance Analysis

For each i, $i = B+1, B+2, \ldots, D$, let $q_i(\mathbf{z})$ or, simply, q_i denote the ratio of posterior error probabilities defined by

$$q_i = \frac{\Pr(E_i = 1 \mid \mathbf{z})}{1 - \Pr(E_i = 1 \mid \mathbf{z})} \quad , \tag{3}$$

evaluated based on a set of parity-check equations Ω_i, where E_i denotes a binary random variable which realization is the noisy bit on the ith codeword bit. In the particular case where all the parity-check equations are orthogonal and involve exactly two unknown bits beside the considered one, and s among $\mid \Omega \mid$ parity-check equations are satisfied, we have

$$q_i = q = \frac{p}{1-p} \left(\frac{1 + (1-2p)^2}{1 - (1-2p)^2} \right)^{|\Omega| - 2s} \quad , \tag{4}$$

recalling that p denotes the crossover probability of the BSC.

The same expression may also be used as an appropriate approximation for nonorthogonal, but linearly independent parity-checks.

The next statements yield the probability that the correct codeword (i.e. the key) appears in the decoding list.

Theorem 1. Let $p \leq 0.5$. The expected probability P_α that the $L-B$ bits with the largest numbers of satisfied parity-checks from a sequence of $D-B$ bits, are all error-free is given by

$$P_\alpha = [1 - \frac{1}{\Sigma_\alpha^{(Pr)}} \sum_{s=|\Omega|-C_\alpha}^{|\Omega|} \Pr(S=s)\ \frac{q}{q+1}]^{L-B}\ , \tag{5}$$

where

$$\Pr(S=s) = p\binom{|\ \Omega\ |}{s} p_w^s (1-p_w)^{|\Omega|-s}\ + (1-p)\binom{|\ \Omega\ |}{s}(1-p_w)^s p_w^{|\Omega|-s}\ , \tag{6}$$

$$p_w = \frac{1-(1-2p)^2}{2}\ , \tag{7}$$

$$\Sigma_\alpha^{(Pr)} = \sum_{s=|\Omega|-C_\alpha}^{|\Omega|} \Pr(S=s)\ , \tag{8}$$

and C_α is the smallest integer such that

$$(D-B)\Sigma_\alpha^{(Pr)} \geq L-B\ . \tag{9}$$

Theorem 2. Let $p \leq 0.5$. The expected probability P_β that the $L-B$ bits with the smallest numbers of satisfied parity-checks from a sequence of $D-B$ bits, are all in error is given by

$$P_\beta = [1 - \frac{1}{\Sigma_\beta^{(Pr)}} \sum_{s=0}^{C_\beta} \Pr(S=s)\ \frac{1}{q+1}]^{L-B}\ , \tag{10}$$

where $\Pr(S=s)$ is given by (6)-(7),

$$\Sigma_\beta^{(Pr)} = \sum_{s=0}^{C_\beta} \Pr(S=s)\ , \tag{11}$$

and C_β is the smallest integer such that

$$(D-B)\Sigma_\beta^{(Pr)} \geq L-B\ . \tag{12}$$

The proofs of Theorem 1 and Theorem 2 follow from the general results given in the Appendix. Note that C_α and C_β are chosen such that the corresponding check sum values occur with sufficiently high probability.

Theorems 1 and 2 imply the following corollary.

Corollary 1. The probability that the correct candidate is on the decoding list is equal to $P_\alpha + P_\beta - P_\alpha P_\beta$, and accordingly it is upper bounded by $\min\{P_\alpha + P_\beta;\ 1\}$ where P_α and P_β are given by Theorems 1 and 2, respectively.

The upper bound of Corollary 1 is expected to be tight when the probability that the correct codeword belongs to both lists remains small compared with both P_α and P_β.

5.2 Complexity Analysis

The search for the desired parity-checks can be done employing a time-memory trade-off as pointed out in [14], [3], [2] and [10].

According to the algorithm structure it can be directly shown that the following two statements hold.

Corollary 2. The time complexity of the pre-processing phase is proportional to $D\binom{N-L}{2}$, without employment of time-memory complexity trade-off, and it is proportional to $D(N-L)$ assuming employment of a sorting based approach and a memory proportional to $(N-L)$.

Corollary 3. Assuming that $|\Omega|$ denotes the average cardinality of the parity-check sets $|\Omega_i|$, and that w denotes the weight of the LFSR characteristic polynomial, the worst case time complexity of the processing phase is proportional to $2^B[\ (D-B)|\Omega| \ +\ (M-L)w\ ]\ mod 2$ additions.

5.3 Illustrative Examples

Tables 1 - 3 illustrate the performance and complexity of the proposed algorithm. Since the dominant operations in the algorithm are $mod 2$ additions, the overall algorithm complexity is proportional to a number of $mod 2$ additions. Also note that $L = 89$ corresponds to a longer LFSR than the longest one ($L = 60$) considered in [10].

6 Comparison

This section gives a comparison of the proposed algorithm and three recently reported ones, [15], [2] and [10], which address the same problem. The main differences in the underlying ideas are pointed out and illustrative examples related to the performance and complexity are presented.

6.1 Comparison of the Approaches

All the fast correlation attacks under comparison are based on certain decoding techniques associated with parity-checks.

The underlying decoding approaches of the algorithms under comparison can be summarized as follows:

Table 1. Proposed algorithm: Noise limit and processing complexity for which the algorithm yields, 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 400000 bits.

Noise Limit	Complexity
0.457	$\sim 2^{38}$ $mod2$ additions
0.460	$\sim 2^{39}$ $mod2$ additions
0.464	$\sim 2^{40}$ $mod2$ additions
0.467	$\sim 2^{41}$ $mod2$ additions
0.469	$\sim 2^{42}$ $mod2$ additions

Table 2. Proposed algorithm: Noise limit and processing complexity for which the algorithm yields, 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 360000 bits.

Noise Limit	Complexity
0.488	$\sim 2^{52}$ $mod2$ additions
0.489	$\sim 2^{53}$ $mod2$ additions
0.490	$\sim 2^{55}$ $mod2$ additions

Table 3. Proposed algorithm - Theoretical analysis: Noise limit, processing complexity and required ample size for which the algorithm yields, with probability close to 1, correct reconstruction of the initial LFSR state, when LFSR length is 89 with an arbitrary primitive characteristic polynomial.

Noise Limit	Complexity	Required Sample
0.469	$\sim 2^{52}$ $mod2$ add.	$\sim 0.25 \cdot 10^{12}$
0.478	$\sim 2^{52}$ $mod2$ add.	$\sim 10^{12}$
0.480	$\sim 2^{52}$ $mod2$ add.	$\sim 4 \cdot 10^{12}$

- one step decoding algorithm (OSDA) [15] is a combination of the threshold decoding [12], and the MDD based on Hamming distance,
- the algorithm [2] is a variant of iterative probabilistic decoding,
- the basic algorithm of [10] is a variant of MDD based on the squared distance measure,
- the proposed algorithm employs list decoding based on the most reliable information sets and MDD with Hamming distance.

The algorithms [15], [10] and the proposed one mainly use parity checks which virtually include only three unknown bits based on employment of exhaustive search, while algorithm [2] uses parity-checks of weight 4 or 5 without employment of exhaustive search in the processing phase.

It appears that list decoding based on the most reliable information sets is more powerful than threshold decoding in conjunction with certain exhaustive search, which explains the gain obtained with the proposed algorithm in comparison with the OSDA algorithms [15]-[16].

The algorithm [2] is based on an iterative probabilistic decoding approach which is a powerful one but complex as well (it requires real number operations as the dominant ones). According to [17], it seems that the iterative decoding techniques can be appropriate for the fast correlation attack when very limited samples are available for processing. So, for a given complexity and a large processing sample, certain noniterative decoding techniques can yield better performance-complexity trade-off.

The approach [10] employs the squared distance measure to avoid an exhaustive search over certain 2^n hypotheses where n is a parameter of the algorithm (see [10] for details) in order to reduce the required exhaustive search. Nevertheless, the use of this squared distance measure rather than Hamming distance becomes suboptimal for MDD over a BSC (see [11], p. 9-10, for example).

The previous discussion points out certain underlying arguments for justifying a better performance-complexity trade-off of the proposed algorithm in comparison with previously reported ones.

6.2 Comparison of Performance and Complexity

The following characteristics of the algorithms are under consideration: performance (noise limit for successful decoding); complexity of the processing; complexity of the preprocessing; required input.

In the considered comparison context, the time complexity of OSDA [15] is proportional to $2^B[\ (L-B)2^{L-B}\binom{N-L}{2}\ +\ (M-L)w\ ]\ mod 2$ additions, where M is length of the sequence for the correlation check, and w denotes the weight of the LFSR characteristic polynomial.

Iterative decoding [2] has expected implementation time complexity proportional to $10Nm(d)$ real number operations assuming at most 10 iterations, where N is the required sample length as well as total number of bits under processing, d is the number of bits involved in a parity-check, and $m(d)$ is the expected number of parity-checks per bit which is approximately equal to $2^{-L}N^{d-1}/(d-1)!$ (see [2]). Also, recall that the algorithm [2] employs *log*-domain processing.

The basic algorithm [10] has time complexity proportional to $2^{2k-L}n\binom{N}{t}$, where k, n, t are the algorithm parameters (see [10]). Also, the following statement is valid: In general, increasing t improves the performance at the cost of an increased precomputation time and increased memory requirement in precomputation, as well as an increased time complexity of the attack itself.

Processing complexity of the proposed algorithm is given by Corollary 3.

Finally, note that the statements on the pre-processing complexity given in [15]-[16], [2] and [10], as well as Definition 1 directly imply that the preprocessing complexity is proportional to $N^\alpha/(\alpha-1)!$ where N is length of the sequence under processing (i.e. required length of the input sequence), and, α is equal to 2 for the algorithms [15]-[17], to t for the algorithm [10], and to the weight

minus one of the employed parity-checks for the algorithm [2]. The previous discussion assumes a straightforward pre-processing without employment of the time-memory trade-off methods.

Tables 4 and 5 present an illustrative comparison of the proposed algorithm and three relevant recently reported algorithms of [15], [2] and [10]. These tables show that the proposed algorithm yields better performance assuming the same input and the same or lower complexity.

Table 4. Comparison of the algorithms, assuming the same inputs: Noise limit and processing complexity 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 360000 bits.

ALGORITHM	Noise Limit	Complexity
FSE2000 [15]	**0.476**	$\sim 2^{52}$
EUROC2000 [2]	**0.482**	$\sim 2^{52}$
Proposed	**0.488**	$\sim 2^{52}$

Table 5. Comparison of the algorithms, assuming the same inputs: Noise limit and processing complexity 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 400000 bits.

ALGORITHM	Noise Limit	Complexity
FSE2000 [15]	**0.420**	$\sim 2^{38}$
CRYPTO2000 [10]	**0.450**	$\sim 2^{38}$
Proposed	**0.457**	$\sim 2^{38}$

Finally, note that an actual implementation of the algorithms strongly depends on the environment and its optimization. Consequently it intentionally will not be discussed here, recalling that as an illustration of this issue, the results reported in [2] and [10] can be considered.

7 Security Consideration of a NESSIE Proposal

This section shows an application of the proposed algorithm for security consideration of a submission to the New European Schemes for Signatures, Integrity and Encryption (NESSIE) Project: a proposal for synchronous stream cipher LILI-128, [19].

The LILI-128 keystream generator is a keystream generator that uses two binary LFSRs, $LFSR_c$ and $LFSR_d$ of lengths 39 and 89, respectively, and two functions f_c and f_d to generate a pseudorandom binary keystream sequence. The components of the keystream generator can be grouped into two subsystems based on the functions they perform; clock control and data generation. The LFSR for the clock-control sub-system is regularly clocked. The output of this subsystem is an integer sequence which controls the clocking of the LFSR within the data-generation subsystem. If regularly clocked, the data-generation subsystem is a simple nonlinearly filtered LFSR (nonlinear filter generator). Accordingly, the LILI-128 generator may be viewed as a clock-controlled nonlinear filter generator with 128 bits secret key.

The state of LILI-128 is defined to be the contents of the two LFSRs. The functions f_c and f_d are evaluated from the current state data, and the feedback bits are calculated. Then the LFSRs are clocked and the keystream bit is output. At initialization, the 128 bit key is used directly to form the initial values of the two shift registers, from left to right, the first 39 bits in $LFSR_c$, then the remaining 89 bits in $LFSR_d$.

Note the following: Assuming a search over 2^{39} hypotheses, cryptanalysis of LILI-128 can be reduced to the problem of LFSR initial state reconstruction with length equal to 89 and output available through a noisy channel.

The underlying assumptions for the security examination of LILI-128 are the following:
- according to the **author's claims** (from the proposal) the noise in the equivalent BSC model corresponds to $p = 0.46875$;
- the available output sequence (keystream) from LILI-128 corresponds to **one hour work**, so that at least $60^3 \cdot 4.8 \cdot 10^6 \sim 10^{12}$ bits are available.

Accordingly, the security evaluation is summarized in Table 6.

Table 6. Theoretical estimations of LILI-128 security level.

ESTIMATION	Complexity of divide and conquer attack
LILI-128 author's claim	$\sim 2^{112}$ operations
estimation based on the proposed algorithm	$\sim 2^{91}$ *mod*2 additions

Note that Table 6 implies that the security margin is reduced by 21 bits in comparison with the claim from the LILI-128 proposal [19].

Finally, it is interesting to compare the developed attack on LILI-128 with its cryptanalysis using the time-memory-data trade-off technique recently reported in [1]. Recall that the trade-off technique [1] is applicable to any keystream generator, and that it is an alternative to the exhaustive key search.

According to [1], an appropriate time-memory-data trade-off when the secret key consists of 128 bits have the following characteristics: (i) required data: $\sim 2^{43}$; (ii) pre-processing complexity: memory $\sim 2^{43}$ and time $\sim 2^{86}$; (iii) processing complexity: memory $\sim 2^{43}$ and time $\sim 2^{86}$ (*recalculation* + *disk read*), where the *recalculation* assumes all operations required for generation 128 LILI-128 output bits. Accordingly, complexity of 2^{86} *recalculations* is proportional to $4 \times 128 \times 2^{86} = 2^{97}$ $mod2$ *additions* (note that factor 4 is due to the employed clock-control operation in LILI-128).

Recall that the cost associated with the disk read operations has to be taken into account, as pointed-out and discussed in [1].

The characteristics of the attacking technique proposed in this paper are the following: (i) required data: $\sim 2^{38}$; (ii) pre-processing complexity: memory $\sim 2^{38}$ and time $\sim 2^{38}$; (iii) processing complexity: memory $\sim 2^{27}$ and time $\sim 2^{91}$ $mod2$ *additions*.

Accordingly, the proposed technique has the following advantages over the time-memory-data trade-off technique [1] in the case of LILI-128 cryptanalysis:
- significantly smaller processing memory;
- significantly smaller overall processing time complexity, particularly due the fact that the proposed approach does not require the expensive disk read operations as a consequence of the required size of processing memory;
- significantly smaller pre-processing complexity;
- shorter data sequence.

8 Conclusions

An improved method for the fast correlation attack on certain stream ciphers has been presented. The proposed algorithm employs two optimal decoding approaches: the list decoding where a candidate is assigned to the list based on the most reliable information sets, and MDD with Hamming distance.

Comparisons show that the proposed algorithm outperforms recently reported algorithms in several scenarios.

A desirable characteristic of the proposed algorithm is its theoretical analyzibility, so that its performance can also be estimated in the cases where the considered experiments are not currently realizable due to technological limitations.

Finally the proposed algorithm can be employed for security examination of the NESSIE proposal for stream cipher LILI-128.

9 Appendix

Elements for the proofs of Theorem 1 and Theorem 2. Assuming that a given number s of satisfied parity-check equations is a realization of a random integer variable S, the Bayes probability of decision error for each bit-i is given by

$$P_B(s) = \min\{\Pr(E = 1 \mid \mathbf{z}), 1 - \Pr(E = 1 \mid \mathbf{z})\}$$

$$= \min\{\Pr(E = 1 \mid S = s), 1 - \Pr(E = 1 \mid S = s)\} = \begin{cases} \frac{q}{1+q} \text{ if } q \leq 1 \ , \\ \frac{1}{1+q} \text{ if } q > 1 \ , \end{cases}$$

and the average Bayes probability of error under the condition Ψ that the random variable S takes values from the set $\{s_0, s_1, ..., s_n\}$ is given by

$$P_B(\Psi) = \sum_s P_B(s) \ \Pr(S = s|\Psi) \ ,$$

where,

$$\Pr(S = s|\Psi) = \frac{\Pr(\Psi|S = s) \Pr(S = s)}{\Pr(\Psi)} \ ,$$

$$\Pr(\Psi|S = s) = \begin{cases} 1, & \text{if } \ S \in \{s_0, s_1, ..., s_n\} \ , \\ 0, & \text{otherwise} \ , \end{cases}$$

$$\Pr(S = s) = p \binom{|\ \Omega\ |}{s} p_w^s (1 - p_w)^{|\Omega|-s} \ + (1 - p) \binom{|\ \Omega\ |}{s} (1 - p_w)^s p_w^{|\Omega|-s} \ ,$$

with $p_w = (1 - (1 - 2p)^2)/2$, and

$$\Pr(\Psi) = \sum_{s:s_0,s_1,...,s_n} \Pr(S = s) \ .$$

Accordingly, after some algebra we obtain each theorem statement.

References

1. A. Biryukov and A. Shamir, "Cryptanalytic Time/ Memory/ Data Tradeoffs for Stream Ciphers", Advances in Cryptology - ASIACRYPT2000, *Lecture Notes in Computer Science*, vol. 1976, pp. 1-13, 2000.
2. A. Canteaut and M. Trabbia, "Improved fast correlation attacks using parity-check equations of weight 4 and 5," Advances in Cryptology - EUROCRYPT'2000, *Lecture Notes in Computer Science*, vol. 1807, pp. 573-588, 2000.
3. V. V. Chepyzhov, T. Johansson and B. Smeets, "A simple algorithm for fast correlation attacks on stream ciphers," Fast Software Encryption - FSE2000, *Lecture Notes in Computer Science*, vol. 1978, pp. 180-195, 2001.
4. P. Elias, "List decoding for noisy channels," *Wescon Convention Record*, Part 2, Institute of Radio Engineers (now IEEE), pp. 94-104, 1957.
5. P. Elias, "Zero error capacity under list decoding," *IEEE Trans. Inform. Theory*, vol 34, pp. 1070-1074, Sept. 1988.
6. P. Elias, "Error-correcting codes for list decoding," *IEEE Trans. Inform. Theory*, vol 37, pp. 5-12, Jan. 1991.
7. 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.

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. T. Johansson and F. Jonsson, "Fast correlation attacks through reconstruction of linear polynomials," Advances in Cryptology - CRYPTO2000, *Lecture Notes in Computer Science*, vol. 1880, pp. 300-315, 2000.
11. S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice Hall, 1983.
12. J. L. Massey, *Threshold Decoding*. Cambridge, MA: MIT Press, 1963.
13. A. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography*. Boca Roton: CRC Press, 1997.
14. W. Meier and O. Staffelbach, "Fast correlation attacks on certain stream ciphers," *Journal of Cryptology*, vol. 1, pp. 159-176, 1989.
15. M. J. Mihaljević, M. P. C. Fossorier and H. Imai, "A low-complexity and high-performance algorithm for the fast correlation attack," Fast Software Encryption - FSE2000, *Lecture Notes in Computer Science*, vol. 1978, pp. 196-212, 2001.
16. M. J. Mihaljević, M. P. C. Fossorier and H. Imai, "An algorithm for cryptanalysis of certain keystream generators suitable for high-speed software and hardware implementations," *IEICE Trans. Fundamentals*, vol. E84-A, pp. 311-318, Jan. 2001.
17. M. J. Mihaljević, M. P. C. Fossorier and H. Imai, "On decoding techniques for cryptanalysis of certain encryption algorithms," *IEICE Trans. Fundamentals*, vol. E84-A, pp. 919-930, Apr. 2001.
18. M. J. Mihaljević and J. Dj. Golić, "A method for convergence analysis of iterative probabilistic decoding," *IEEE Trans. Inform. Theory*, vol. 46, pp. 2206-2211, Sept. 2000.
19. NESSIE list of Accepted Submissions: Proposal for Synchronous Stream Cipher LILI-128, Nov. 2000, http://www.cosic.esat.kuleuven.ac.be/nessie.
20. T. Siegenthaler, "Decrypting a class of stream ciphers using ciphertext only," *IEEE Trans. Comput.*, vol. C-34, pp. 81-85, 1985.
21. J. M. Wozencraft,"List decoding," *Quarterly Progress Report*, vol. 48, pp. 90-95, Research Laboratory of Electronics, MIT, Jan. 15, 1958.
22. 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.

Bias in the LEVIATHAN Stream Cipher

Paul Crowley[1⋆] and Stefan Lucks[2⋆⋆]

[1] cryptolabs Amsterdam
paul@cryptolabs.org
[2] University of Mannheim
lucks@weisskugel.informatik.uni-mannheim.de

Abstract. We show two methods of distinguishing the LEVIATHAN stream cipher from a random stream using 2^{36} bytes of output and proportional effort; both arise from compression within the cipher. The first models the cipher as two random functions in sequence, and shows that the probability of a collision in 64-bit output blocks is doubled as a result; the second shows artifacts where the same inputs are presented to the key-dependent S-boxes in the final stage of the cipher for two successive outputs. Both distinguishers are demonstrated with experiments on a reduced variant of the cipher.

1 Introduction

LEVIATHAN [5] is a stream cipher proposed by David McGrew and Scott Fluhrer for the NESSIE project [6]. Like most stream ciphers, it maps a key onto a pseudorandom keystream that can be XORed with the plaintext to generate the ciphertext. But it is unusual in that the keystream need not be generated in strict order from byte 0 onwards; arbitrary ranges of the keystream may be generated efficiently without the cost of generating and discarding all prior values. In other words, the keystream is "seekable". This property allows data from any part of a large encrypted file to be retrieved efficiently, without decrypting the whole file prior to the desired point; it is also useful for applications such as IPsec [2]. Other stream ciphers with this property include block ciphers in CTR mode [3]. LEVIATHAN draws ideas from the stream ciphers WAKE [9] and SEAL [7], and the GGM pseudo-random function (PRF) construction [1].

The keystream is bounded at 2^{50} bytes. Though the security goals are stated in terms of key recovery, it is desirable that distinguishing this keystream from a random binary string should be as computationally expensive as an exhaustive search of the 128 or 256-bit keyspace. Keystream generation is best modelled as a key-dependent function $\mathrm{Lev} : \{0,1\}^{48} \mapsto \{0,1\}^{32}$, mapping a location in the stream to a 32-bit output word; catenating consecutive values of this function from 0 gives the entire keystream:

$$\mathrm{Lev}(0)|\mathrm{Lev}(1)|\mathrm{Lev}(2)|\ldots|\mathrm{Lev}(2^{48}-1)$$

⋆ This research was supported by convergence integrated media GmbH
⋆⋆ This research was supported by Deutsche Forschungsgemeinschaft (DFG) grant Kr 1521/2

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 211–218, 2002.

Finding $\mathrm{Lev}(i)$ for arbitrary i is not especially fast. However, once this is done, intermediate values can usually be reused to find $\mathrm{Lev}(i+1), \mathrm{Lev}(i+2)\ldots$ much more efficiently. This is because the internal structure of the cipher is based on a forest of 2^{32} binary trees, each of which generates 2^{16} words of output, as shown in Figure 1.

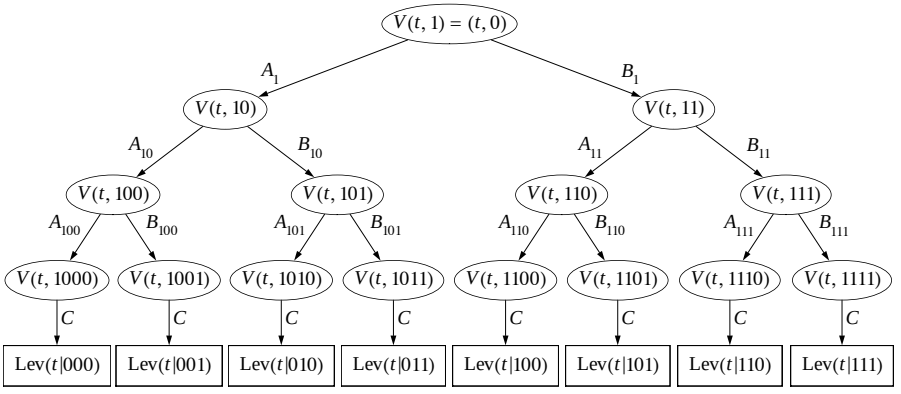


Fig. 1. Computation of an entire output tree of 8 words with $h = 3$. In the full cipher, $h = 16$ and the complete output is built from 2^{32} such trees.

The notation we use to specify this function precisely is somewhat different from that given in [5], but is convenient for our purposes; we treat z as a parameter, rather than as a word of state. The cipher is parameterised on n and h, where n is divisible by 4 and $n \geq h$; LEVIATHAN sets $n = 32$ and $h = 16$. $|$ denotes catenation of bit strings, $\overline{x}$ bitwise complementation of x, $\oplus$ the XOR operation (addition in Z_2^n or $Z_2^{n/4}$ as appropriate), and $+$ addition in the group Z_{2^n}, treating the first bit of the bitstring as the most significant and padding bitstrings shorter than n bits with zeroes on the left. We specify the forest structure illustrated in Figure 1 recursively:

$$\begin{aligned}
\mathrm{Lev} : \{0,1\}^{n+h} &\mapsto \{0,1\}^n \\
\mathrm{Lev}(t|z) &= C(V(t,1|z)) \qquad (|t| = n, |z| = h) \\
V(t,1) &= (t,0) \\
V(t,z|0) &= A_z(V(t,z)) \\
V(t,z|1) &= B_z(V(t,z))
\end{aligned}$$

The internal state that functions A, B, and C operate on (and the functions D, F, G used to define them) is a 2-tuple of bitstrings (x, y); we treat this as distinct from the catenated bitstring $x|y$. The functions L, R, and S operate on bytes within a word: L and R are rotates, while S provides nonlinearity with the key-dependent permutations $S_{0\ldots3}$ which map $\{0,1\}^{n/4}$ onto itself. These

permutations are generated by the key schedule, which we omit. Note that F and G operate on each word of the tuple independently; mixing is provided by D.

$$\begin{aligned}
C(x,y) &= x \oplus y \\
A_z &= F \circ D_z \\
B_z &= G \circ D_z \\
D_z(x,y) &= (2x+y+2z, x+y+z) \\
F(x,y) &= (L(S(L(S(x)))), S(R(S(R(y))))) \\
G(x,y) &= (S(R(S(R(\overline{x})))), L(S(L(S(y))))) \\
L(x_3|x_2|x_1|x_0) &= x_2|x_1|x_0|x_3 \qquad (|x_3| = |x_2| = |x_1| = |x_0| = n/4) \\
R(x_3|x_2|x_1|x_0) &= x_0|x_3|x_2|x_1 \\
S(x_3|x_2|x_1|x_0) &= x_3 \oplus S_3(x_0)|x_2 \oplus S_2(x_0)|x_1 \oplus S_1(x_0)|S_0(x_0)
\end{aligned}$$

[5] gives a functionally different definition of D ($D_z(x,y) = (2x+y+z, x+y+z)$); the one given here is that intended by the authors [4] and used to generate the test vectors, though the difference is not relevant for our analysis.

We present two biases in the LEVIATHAN keystream that distinguish it from a random bit string. We know of no other attacks against LEVIATHAN more efficient than brute force.

2 PRF-PRF Attack

Both attacks focus on consecutive pairs of outputs generated by $\mathrm{LevPair}(i) = (\mathrm{Lev}(i|0), \mathrm{Lev}(i|1))$. Clearly, LevPair generates the same 2^{50}-byte keystream as Lev, so a distinguisher for one is a distinguisher for the other. Such pairs are interesting because they are the most closely related outputs in the tree structure; [5] refers to attacks using such pairs as "up-and-down attacks". We can expand the formula for LevPair as follows:

$$\begin{aligned}
\mathrm{LevPair}(t|z) &= (\mathrm{Lev}(t|z|0), \mathrm{Lev}(t|z|1)) \\
&= (C(V(t,1|z|0)), C(V(t,1|z|1))) \\
&= (C(F(D_{1|z}(V(t,1|z)))), C(G(D_{1|z}(V(t,1|z))))))
\end{aligned}$$

From this we define functions LevAbove which generates the last common ancestor of such an output pair as illustrated in Figure 2, and PairCom which generates the output pair from the ancestor:

$$\begin{aligned}
\mathrm{LevAbove}(t|z) &= D_{1|z}(V(t,1|z)) \qquad (|z| = h-1) \\
\mathrm{PairCom}(x,y) &= (C(F(x,y)), C(G(x,y)))
\end{aligned}$$

from which we can see $\mathrm{LevPair} = \mathrm{PairCom} \circ \mathrm{LevAbove}$ as stated. We model LevAbove as a random function throughout, and focus on the properties of PairCom.

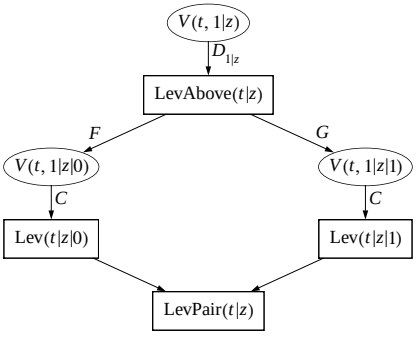


Fig. 2. Final stage of LevPair output; LevAbove finds the last common ancestor of the pair.

This structure gives us our first distinguisher. Though PairCom has the same domain as range, it is not in general bijective; it can be modelled more accurately as a random function. Thus a collision can occur in LevPair, given two distinct inputs, if there is a collision either in LevAbove or in PairCom, and if we model both as random functions the probability of an output collision for two random distinct inputs to LevPair is thus approximately $2^{-2n} + (1 - 2^{-2n})2^{-2n} \approx 2^{1-2n}$, twice what it should be if the keystream were a random binary string.

For $n = 32$, this increased probability of collisions between output word pairs can be observed with a birthday attack after around 2^{33} output pairs (2^{36} bytes) have been generated; the techniques of [8] may be used to reduce the memory demands of this attack, though this slows the attack by a factor of approximately $(h+1)/4 = 4.25$ where h is the height of the tree, since probes can no longer take advantage of the higher efficiency of sampling consecutive values of LevPair.

3 S-Box Matching Attack

The definitions of the F and G functions are very similar; G is the same as F except that it treats its inputs in the opposite order, and inverts one of them. If G did not apply bitwise inversion to its first input (call this function G'), then the two functions would be related by $F \circ \mathrm{Swap} = \mathrm{Swap} \circ G'$ (with Swap having the obvious definition $\mathrm{Swap}(x, y) = (y, x)$); this would mean in turn that $F(a, a) = \mathrm{Swap}(G'(a, a))$ for any a, and thus that $C(F(a, a)) = C(G'(a, a))$, with the result, as we shall see, that repeating pairs were visible in the output roughly twice as often as they should be. The inversion on the first input of G breaks this symmetry; however, it turns out that it does not prevent a related attack.

Computation of PairCom requires 32 S-box lookups, but for each computation of the S function the same 8-bit index, drawn from the least significant byte, is fed to each of the four S-boxes. Changes to the other bytes carry directly into the output of S, without nonlinearity or mixing; in other words,

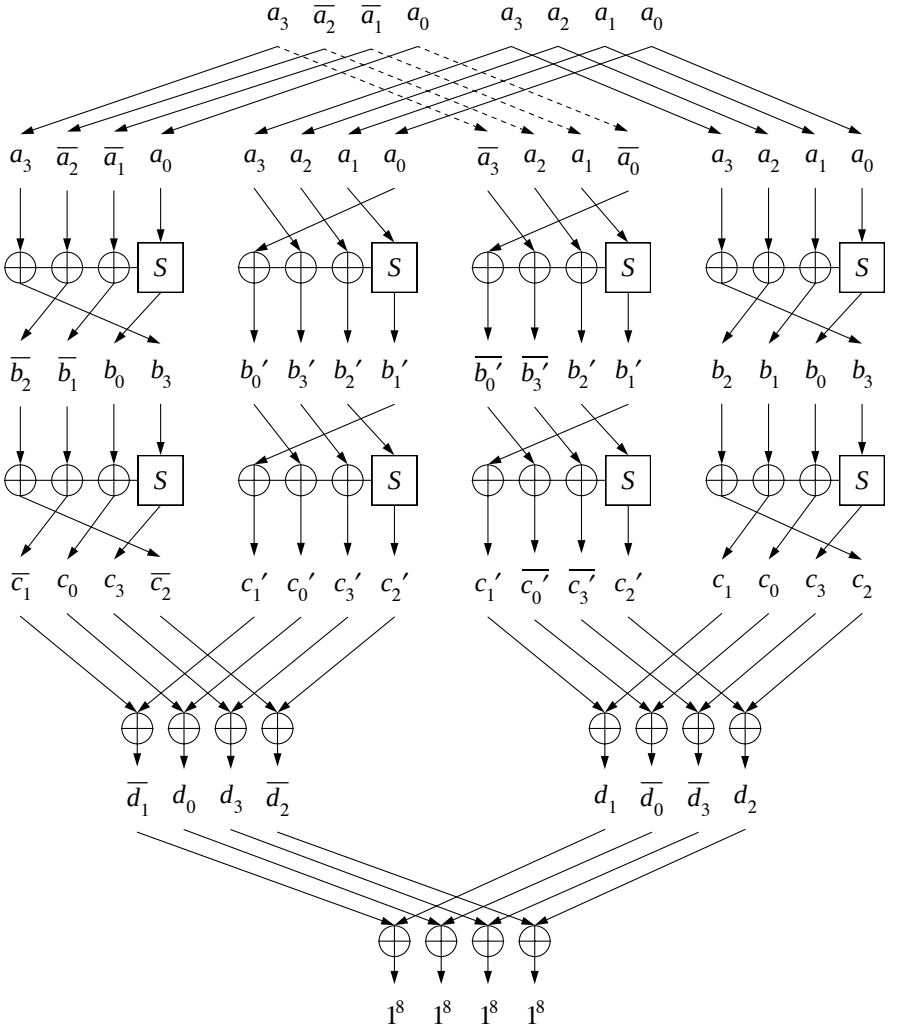


Fig. 3. S-box matching input to $C \circ \text{PairCom}$. The function F is on the left, G on the right, and C underneath; dotted lines indicate bitwise inversion (the first step of the G function) and the "$\oplus \oplus \oplus S$" symbol represents the function $S(x_3|x_2|x_1|x_0) = x_3 \oplus S_3(x_0)|x_2 \oplus S_2(x_0)|x_1 \oplus S_1(x_0)|S_0(x_0)$.

where $\Delta x = \Delta x_3|\Delta x_2|\Delta x_1|0^{n/4}$, we find $S(x \oplus \Delta x) = S(x) \oplus \Delta x$. We call this least significant byte the *index* to the S-box. If (x, y) is the input to PairCom, only bytes x_3, x_0 of x are indices to S-boxes in F, and only bytes x_2, x_1 are indices in G; by inverting only these two bytes in our pair (a, a), we can avoid the symmetry-breaking effect of the inversion as far as the nonlinear components are concerned, which results in the same four S-box indices being used in both the F and G branches of PairCom.

Figure 3 illustrates this attack. For an arbitrary n-bit string $a = a_3|a_2|a_1|a_0$, we define symbols for intermediate values in $F(a, a)$:

$$\begin{aligned}
b_3|b_2|b_1|b_0 &= S(a_3|a_2|a_1|a_0) \\
b'_0|b'_3|b'_2|b'_1 &= S(a_0|a_3|a_2|a_1) \\
c_2|c_1|c_0|c_3 &= S(b_2|b_1|b_0|b_3) \\
c'_1|c'_0|c'_3|c'_2 &= S(b'_1|b'_0|b'_3|b'_2) \\
d_3|d_2|d_1|d_0 &= (c_3|c_2|c_1|c_0) \oplus (c'_3|c'_2|c'_1|c'_0)
\end{aligned}$$

With these definitions, we find that $\mathrm{PairCom}(a_3|\overline{a_2}|\overline{a_1}|a_0,\ a_3|a_2|a_1|a_0) = (\overline{d_1}|d_0|d_3|\overline{d_2},\ d_1|\overline{d_0}|\overline{d_3}|d_2)$:

$$\begin{aligned}
C(F(x,y)) &= C(L(S(L(S(x)))),\ S(R(S(R(y))))) \\
&= C(L(S(L(S(a_3|\overline{a_2}|\overline{a_1}|a_0)))),\ S(R(S(R(a_3|a_2|a_1|a_0))))) \\
&= C(L(S(L(b_3|\overline{b_2}|\overline{b_1}|b_0))),\ S(R(S(a_0|a_3|a_2|a_1)))) \\
&= C(L(S(\overline{b_2}|\overline{b_1}|b_0|b_3)),\ S(R(b'_0|b'_3|b'_2|b'_1))) \\
&= C(L(\overline{c_2}|\overline{c_1}|c_0|c_3),\ S(b'_1|b'_0|b'_3|b'_2)) \\
&= C(\overline{c_1}|c_0|c_3|\overline{c_2},\ c'_1|c'_0|c'_3|c'_2) \\
&= \overline{d_1}|d_0|d_3|\overline{d_2} \\
C(G(x,y)) &= C(S(R(S(R(\overline{x})))),\ L(S(L(S(y))))) \\
&= C(S(R(S(R(\overline{a_3}|a_2|a_1|\overline{a_0})))),\ L(S(L(S(a_3|a_2|a_1|a_0))))) \\
&= C(S(R(S(\overline{a_0}|\overline{a_3}|a_2|a_1))),\ L(S(L(b_3|b_2|b_1|b_0)))) \\
&= C(S(R(\overline{b'_0}|\overline{b'_3}|b'_2|b'_1)),\ L(S(b_2|b_1|b_0|b_3))) \\
&= C(S(b'_1|\overline{b'_0}|\overline{b'_3}|b'_2),\ L(c_2|c_1|c_0|c_3)) \\
&= C(c'_1|\overline{c'_0}|\overline{c'_3}|c'_2,\ c_1|c_0|c_3|c_2) \\
&= d_1|\overline{d_0}|\overline{d_3}|d_2
\end{aligned}$$

From this it is clear that for any input of the appropriate form, one output word is the inverse of the other; or in other words, if we now XOR the two word outputs from PairCom together (which, conveniently, is the same as applying the LEVIATHAN compression function C a second time), we find

$$C(\mathrm{PairCom}(a_3|\overline{a_2}|\overline{a_1}|a_0,\ a_3|a_2|a_1|a_0)) = \overline{d_1}|d_0|d_3|\overline{d_2} \oplus d_1|\overline{d_0}|\overline{d_3}|d_2 = 1^n$$

for *all* values of $a_{3\ldots0}$.

Since we model LevAbove as a random function we conclude that inputs to PairCom have probability 2^{-n} of matching this form in the normal calculation of LevPair. Where inputs do not match this form, we assume that PairCom behaves as a random function and therefore that for random (x, y) not matching this form, $\Pr\left(C(\mathrm{PairCom}(x, y)) = 1^n\right) = 2^{-n}$; this assumption is borne out by experiment. From this we conclude that LevPair is twice as likely as a random function to produce an output (x_0, x_1) such that $C(x_0, x_1) = 1^n$

$$\Pr\left(C(\mathrm{LevPair}(t|z)) = 1^n\right) = 2^{-n} + (1 - 2^{-n})2^{-n} \approx 2^{1-n}$$

which in turn implies that 64-bit aligned segments of keystream of this form are twice as frequent as they should be, yielding another distinguisher.

For $n = 32$, a test for the presence of this bias should therefore take on the order of 2^{33} samples of LevPair, ie 2^{36} bytes, as for the previous attack.

4 Experiments

We looked for these biases on a reduced version of LEVIATHAN with $n = 16, h = 16$.

For the PRF-PRF attack, we ran over 256 distinct keys generating $N = 6291456$ 32-bit LevPair outputs for each, and sorting them to find collisions. We count as a collision each instance where a distinct pair of inputs result in the same output; thus, where $m > 2$ outputs have the same value, we count this as $m(m-1)/2$ distinct collisions. For a random function we would expect to find approximately[1] $256(N(N-1)/2)/2^{2n} \approx 1179678$ collisions in total across all keys, while the PRF-PRF attack would predict an expected $256(N(N-1)/2)/2^{2n-1} \approx 2359296$. The experiment found 2350336 collisions; this is 1077.9 standard deviations (SDs) from the expected value in the random function model, and 5.83 SDs from the expected value in the model provided by the PRF-PRF attack. This shows that this model identifies a substantial bias in the cipher, but there is a further bias in the collision probability of roughly 0.38% yet to be accounted for.

For the S-box matching attack, we generated $N = 16777216$ LevPair outputs for each of 256 keys, counting outputs with the $C(x, y) = 1^{32}$ property. A random function would generate an expected $256N/2^{16} = 65536$ such outputs, while the S-box matching attack predicts that LevPair will generate an expected $256N/2^{15} = 131072$ such outputs. The experiment found 135872 such outputs; this is 274.8 SDs from the expected value in the random function model, and 13.26 SDs from the expected value in the model provided by the S-box matching attack. Again, this shows that while a substantial source of bias has been identified, there is still a bias of 3.66% yet to be accounted for. Scott Fluhrer has reported finding this attack effective in experiments against the full LEVIATHAN with $n = 32, h = 16$.

5 Conclusions

We have shown two forms of bias in the output of the LEVIATHAN keystream generator, either of which distinguish it from a random function with 2^{36} known bytes of output; we have not as yet found a way to recover key material using these distinguishers. These distinguishers can both be applied to the same

[1] The approximation $E\left(|\{\{x, y\} : f(x) = f(y)\}|\right) \approx |A|\left(|A| - 1\right)/2\,|B|$ for the number of collisions in a random function $f : A \mapsto B$ is very precise where $|B|$ is large; where we refer to the predictions of the random function model, it is the model with this approximation.

portion of keystream for greater statistical significance. Both make use of compression in the PairCom function.

Despite these attacks, LEVIATHAN demonstrates that a tree-based cipher could offer many advantages. It is to be hoped that similar designs, offering the same speed and flexibility but resistant to this and other attacks, will be forthcoming.

Acknowledgements. Thanks to Rüdiger Weis for helpful commentary and suggestions, and to the LEVIATHAN authors for providing an implementation of the first experiment and for useful discussion.

References

1. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986.
2. IP security protocol (ipsec). `http://www.ietf.org/html.charters/ipsec-charter.html`.
3. Helger Lipmaa, Philip Rogaway, and David Wagner. Comments to NIST concerning AES modes of operation: CTR-mode encryption, 2000.
4. David A. McGrew. Re: Possible problems with leviathan? Personal email, November 2000.
5. David A. McGrew and Scott R. Fluhrer. The stream cipher LEVIATHAN. NESSIE project submission, October 2000.
6. NESSIE: New European schemes for signatures, integrity, and encryption. `http://www.cryptonessie.org/`.
7. Phillip Rogaway and Don Coppersmith. A software-optimized encryption algorithm. In Ross Anderson, editor, *Fast Software Encryption*, pages 56–63. Springer-Verlag, 1994.
8. Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.
9. David Wheeler. A bulk data encryption algorithm. In 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.

URL for this paper: `http://www.ciphergoth.org/crypto/leviathan`

Analysis of SSC2

Daniel Bleichenbacher[1] and Willi Meier[2]

[1] Bell Laboratories, Rm. 2A-366, 700 Mountain Av
Murray Hill, NJ 07974-0636, USA
bleichen@bell-labs.com
[2] FH Aargau
CH-5210 Windisch
meierw@fh-aargau.ch

Abstract. This paper analyses the stream cipher SSC2 [ZCC00]. We describe some weaknesses and attacks exploiting these weaknesses. The strongest attack needs about 2^{52} words of known key stream and has a time complexity of about 2^{75}.

1 Introduction

The stream cipher SSC2 has been proposed in [ZCC00]. It consists of a filter generator and a filtered lagged Fibonacci generator. The outputs of the two generators are XORed to give the key stream.

This paper investigates the strength of the stream cipher SSC2. We describe some weaknesses and some attacks based on those weaknesses. The strongest attack needs about 2^{52} words of known key stream and has a time complexity of about 2^{75}. Note, recently Hawkes, Quick and Rose have found a new attack that is faster than ours [HR01].

The outline of the paper is as follows. Section 3 analyses the linear feedback shift register. In Section 4 properties of the nonlinear filter are described. We compute the distribution of some carry bits, so that we can improve one of the attacks later.

In Section 6 properties of the lagged Fibonacci generator are discussed. Then we describe two different attacks on SSC2, exploiting different weaknesses of SSC2. Our first attack needs about 2^{52} known plaintext and 2^{75} time. This attack is divided into two sections. First, in Section 7 we describe how to find the internal state of the LFSR by exploiting the short period of the lagged Fibonacci generator. Then in Section 8 we cryptanalyse the lagged Fibonacci generator. Section 9 describes our second attack exploiting the bias in the lagged Fibonacci generator. One variant of this attack needs 2^{32} words of known plaintext, but has a time complexity of 2^{123}. In Section 10 we identify a bug in the frame key generation.

2 The Structure of SSC2

The stream cipher SSC2 as shown in figure 1 consists of a filter generator and a lagged Fibonacci generator. The word-oriented LFSR has 4 stages with each

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 219–232, 2002.

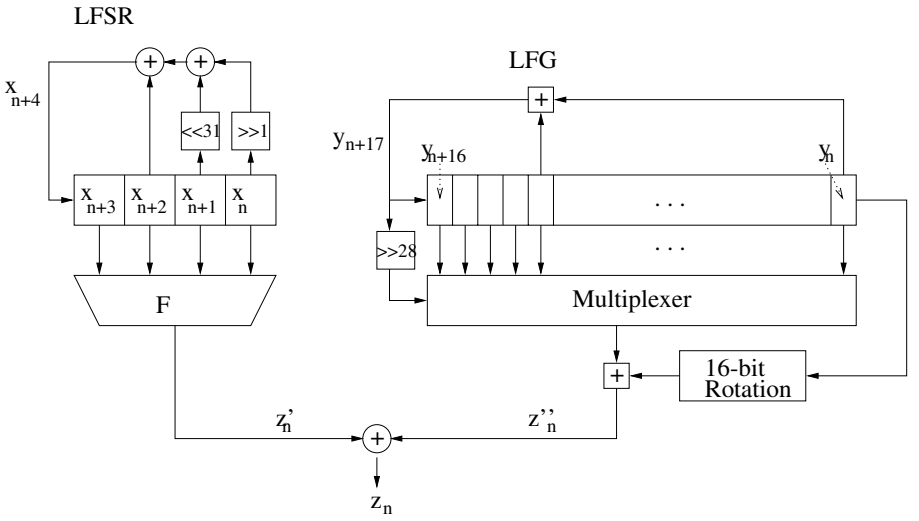


Fig. 1. The key stream generator of SSC2.

stage containing one word. It generates a new word and shifts out an old word at every clock. The nonlinear filter has the 4-word content of the LFSR as its input and a single word as output. 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 to give a word of the key stream.

3 The Linear Feedback Shift Register

The LFSR in [ZCC00] is described using 4 registers where each of them is 32 bits long. The LFSR is regularly clocked. Clocking the LFSR turns a state $(x_{n+3}, x_{n+2}, x_{n+1}, x_n)$ into the state $(x_{n+4}, x_{n+3}, x_{n+2}, x_{n+1})$, where the 32-bit variables satisfy the linear recurrence relation

$$x_{n+4} = x_{n+2} \oplus (x_{n+1} \ll 31) \oplus (x_n \gg 1). \tag{1}$$

More convenient for this paper is another equivalent description of the LFSR using 8 registers where each of them is 16 bits long as shown in figure 2.

The linear feedback shift register can be described using 8 registers of 16 bit each. Cycling this register twice is equivalent to cycling the 32-bit register once. It can be observed that the least significant bits of the register determine the least significant bits of further states of the register. If we know the t least significant bits of $\mathrm{hi}(x_i), \mathrm{lo}(x_i)$ for all $n+1 \le i \le n+4$ then we can determine, the t least significant bits of x_n (except the bit that is not used in the nonlinear filter). For $v < t$ we can further determine the $t-v$ least significant bits of $\mathrm{hi}(x_i)$ and $\mathrm{lo}(x_i)$ for all $n+4v+1 \le i \le n+4v+4$.

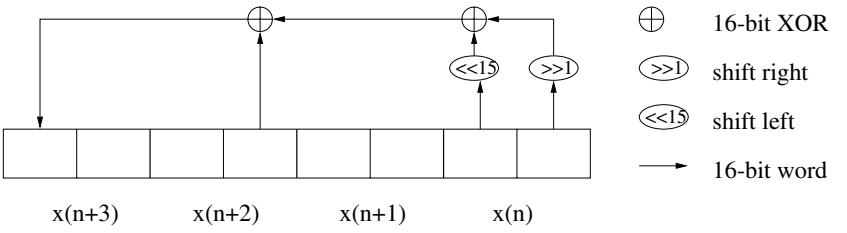


Fig. 2. The linear feedback shift register using 16-bit operations. One cycle in the 32-bit variant corresponds to two cycles in the 16-bit variant. It can be observed that the low significant bits of this register depend on low significant bits of previous states.

4 The Nonlinear Filter

SSC2 derives the sequence z'_n from the LFSR state $x_{n+3}, x_{n+2}, x_{n+1}, x_n$ by using a nonlinear filter F. The filter uses 16-bit rotation and therefore describing F using 16-bit arithmetic rather than 32-bit simplifies the analysis. Let $\mathrm{hi}(x)$ and $\mathrm{lo}(x)$ denote the high order 16 bits rsp. the low order 16 bits of a 32 bit integer. The original description of the nonlinear filter used 16-bit rotations, which we will denote by $(x \ggg 16)$. However, we can get rid of these rotations by describing the nonlinear filter using 16-bit block rather than 32-bit block. The resulting description of the nonlinear filter is given in Figure 3. Additional carry bits $c1, c3$ and $c5$ are necessary to simulate 32-bit additions with 16-bit operations.

Correlation Properties. Ideally, correlations of the output of a nonlinear filter to a linear function of the LFSR should be minimized. The nonlinear filter of SSC2 has some correlations. At the moment we do not know, whether such correlations can be used in an attack.

Property 1. *The nonlinear filter of SSC2 satisfies the following equation, which is a linear approximation.*

$$Prob\left(\mathrm{lsb}(\mathrm{hi}(z'_n)) = \mathrm{lsb}(x_{n+3}) \oplus \mathrm{lsb}(\mathrm{hi}(x_{n+1}))\right) = 7/12$$

The probability $7/12$ was estimated using heuristic assumptions about the independence of the inputs and was verified experimentally.

Philip Hawkes, Frank Quick and Greg Rose have recently presented some other correlation properties in [HR01]. They use these observations for a new attack that is faster than ours.

5 Approximation to the Nonlinear Filter

In the analysis we will use an approximation G to the nonlinear filter F. Essentially G tries to guess the carry bits, during the computation of F. The motivation for approximating F by G is the simplification that we get, because there are less dependencies in G. The disadvantage of using an approximation G,

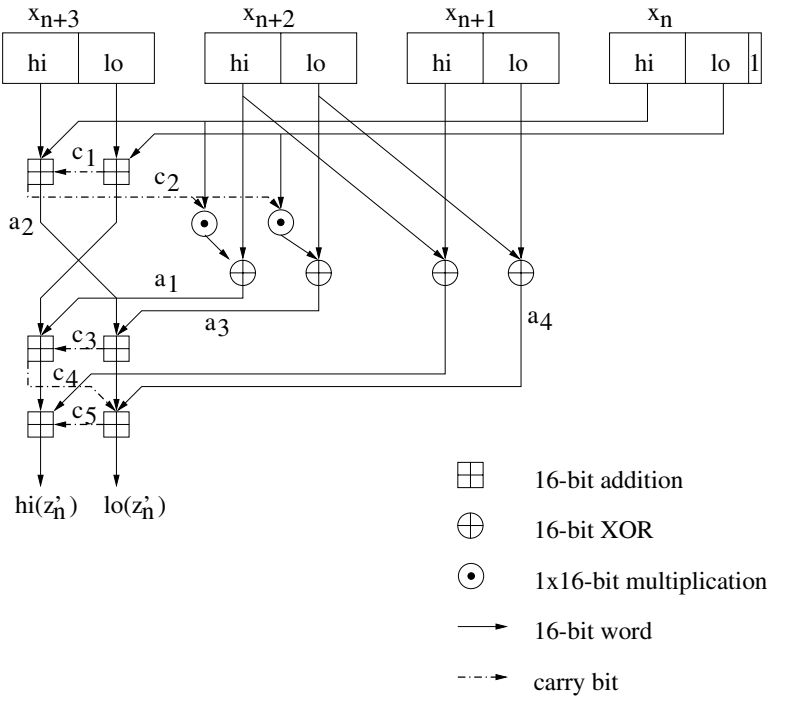


Fig. 3. This figure describes the nonlinear filter F of SSC2. Here, we give an equivalent description of the filter using 16-bit operations rather than the original 32-bit operations. This change is provided to simplify further analysis of SSC2, because in our description does not need any rotations of integers. Note, that this description needs 3 additional carry bits to simulate the 32-bit additions.

rather than F itself is the error that is introduced. In our attack we will choose sequences of key stream, and hope that the differentials between G and F cancel out. Therefore we will have to repeat our attack with different parts of the key stream. In this section we describe the function G and compute the probability distribution of the error $F \oplus G$.

The function G is described in Figure 4. The difference to F is that in G no carry bits are computed. Rather the effect of carry bits in F is simulated in G by the additional inputs $d_1 \in \{0, 1, 2\}, d_2 \in \{0, 1\}$ and $d_3 \in \{0, 1, 2\}$. Otherwise, F and G are identical. In particular, let c_1, c_2, c_3, c_4 and c_5 be the carry bits during the computation of $F(x_{n+3}, x_{n+2}, x_{n+1}, x_n)$ and let $d_1 = c_1 + c_4, d_2 = c_2$ and $d_3 = c_3 + c_5$. Then

$$F(x_{n+3}, x_{n+2}, x_{n+1}, x_n) = G(x_{n+3}, x_{n+2}, x_{n+1}, x_n, d_1, d_2, d_3).$$

There are two possible approaches for our attack. We could either try all possible combinations for the values d_1, d_2, d_3 or we can guess the most likely values and then try our attack on different sequences of the key stream until we

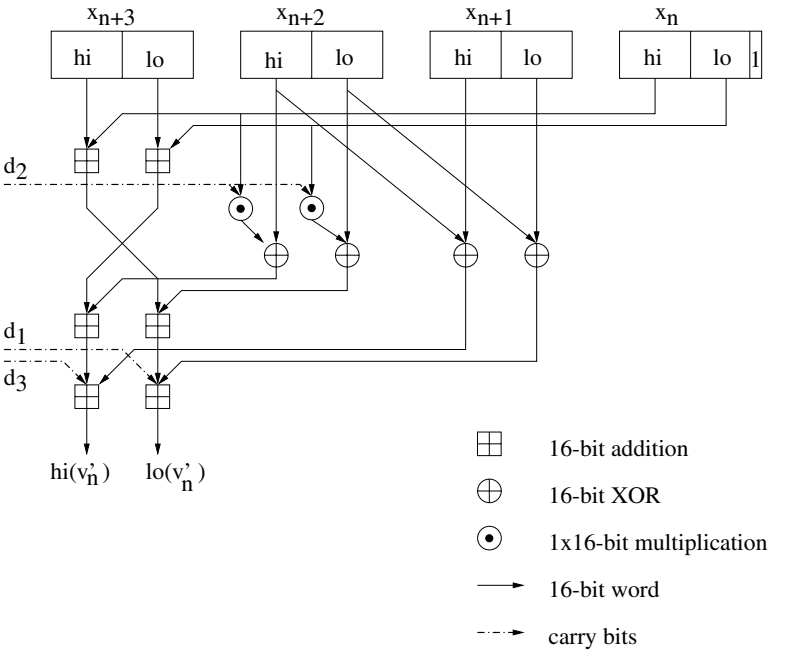


Fig. 4. A picture of the function $G(x_{n+3}, x_{n+2}, x_{n+1}, x_n, d_1, d_2, d_3)$, which is used to approximate F. G and F would be equivalent if the carry bits $d_2 = c_2$, $d_1 = c_1 + c_4$ and $d_3 = c_3 + c_5$ are used. However, we will use $d_1 = 1$ and $d_3 = 1$ and compute the distribution of the differential.

have found a sequence where our guess was correct. We will choose the second approach, because it is slightly more efficient. We will use $d_1 = d_3 = 1$ and only guess d_2. We are therefore interested in the differential

$$\Delta_n = F(x_{n+3}, x_{n+2}, x_{n+1}, x_n) \oplus G(x_{n+3}, x_{n+2}, x_{n+1}, x_n, 1, c_2, 1).$$

To analyse Δ_n we first investigate the distribution of $c_1 + c_4$ and $c_3 + c_5$ in the computation of F. Let

$$\begin{aligned} a_1 &= (c_2 \cdot \mathrm{hi}(x_n)) \oplus \mathrm{hi}(x_{n+2}) \\ a_2 &= \mathrm{hi}(x_{n+3}) + \mathrm{hi}(x_n) \bmod 2^{16} \\ a_3 &= (c_2 \cdot \mathrm{lo}(x_n)) \oplus \mathrm{lo}(x_{n+2}) \\ a_4 &= \mathrm{lo}(x_{n+1}) \oplus \mathrm{lo}(x_{n+2}). \end{aligned}$$

Then it can be observed that

$$c_1 + c_4 = \left\lfloor \frac{\mathrm{lo}(x_{n+3}) + \mathrm{lo}(x_n) + a_1}{2^{16}} \right\rfloor$$

and

$$c_3 + c_5 = \left\lfloor \frac{a_2 + a_3 + a_4 + c_4}{2^{16}} \right\rfloor$$

The expression for c_1+c_4 and c_3+c_5 are basically (ignoring the carry bit c_4) sums of three 16-bit integers. In [SM91, Section 2.2] the carry of sums of three integers has been analysed. Under the assumption that the summands are independent and uniformly distributed it follows that the probability of $c_1 + c_4 = 0$ and $c_1 + c_4 = 2$ is $\approx 1/6$ in each case, and the probability of $c_1 + c_4 = 1$ is $\approx 2/3$. $c_3 + c_5$ is similarly distributed.

We now compute the distribution of $\mathrm{hi}(\Delta_n)$ under the assumption that $d_2 = c_2$. Hereby, we have to distinguish two cases. Firstly, if $c_3+c_5 = 1$, then $\mathrm{hi}(\Delta_n) = 0$. Secondly, if $c_3 + c_5 \neq 1$ then $\mathrm{hi}(\Delta_n)$ is the XOR of two 16-bit integers with difference 1. Hence the distribution of $\mathrm{hi}(\Delta_n)$ is as follows

$\mathrm{hi}(\Delta_n)$	Probability
0	$\approx 2/3$
1	$\approx 1/6$
$2^k - 1$ for $k = 2, \ldots, 15$	$\approx 2^{-k}/3$
$2^{16} - 1$	$\approx 2^{-15}/3$

Similar arguments can be used to show that $\mathrm{lo}(\Delta_n)$ has the same distribution as $\mathrm{hi}(\Delta_n)$.

6 Lagged Fibonacci Generator

From [ZCC00] we briefly recall the description of the component of SSC2 consisting of a (suitably filtered) lagged Fibonacci generator. The generator is word oriented and is characterized by the linear recurrence relation

$$y_{n+17} = y_{n+12} + y_n \pmod{2^{32}}.$$

Since the corresponding feedback polynomial $x^{17} + x^5 + 1$ is irreducible over $\mathbb{Z}/(2)$ it follows that the sequence has a period of $(2^{17} - 1)2^{31}$ (or a divisor of this number and it achieves the maximal period if one of the least significant bits of $f_1, \ldots, f_{17}$ is 1). In SSC2 this generator is implemented with a 17-stage circular buffer B and two pointers s and r. Initially $B[17]$, $B[16]$,..., $B[1]$ are loaded with $y_0, y_1, ...y_{16}$, and s and r are set to 5 and 17, respectively. At every clock, a new word is computed by taking the sum of $B[r]$ and $B[s]$ mod 2^{32}. The word $B[r]$ is then replaced by the new word, and the pointers r and s are decreased by 1. A filter is used to produce the output. The output word z''_n is computed from the replaced word y_n and another word selected from the buffer B. The selection uses a multiplexer based on the most significant 4 bits of the newly generated word y_{n+17}. The output word z''_n is given by

$$z''_n = (y_n \ggg 16) + B[1 + ((y_{n+17} \gg 28) + s_{n+1} \bmod 16)] \bmod 2^{32}, \qquad (2)$$

where s_{n+1} denotes the value of s at time $n+1$ (note that $1 \leq s \leq 17$). Observe that this filter is not memoryless and has indeed a period 17, which has to be considered when computing the period of the z''_n. In [ZCC00] it is shown that

Property 2. *The period of* z''_n *divides* $17(2^{17}-1)2^{31}$.

The multiplexer can select the same elements of the Fibonacci generator several times in a row. The linear properties of the Fibonacci generator are only partially destroyed. It can be observed that the sequence S_n defined by

$$S_n = z''_{n+17} - z''_{n+12} - z''_n \bmod 2^{32}$$

has a nonuniform distribution. This distribution has the following property

Property 3. *Let*

$$\mathcal{A} = \{\texttt{0x0000000}, \texttt{0x00000001}, \texttt{0xffff0000}, \texttt{0xffff0001}\}.$$

Then $S_n \in \mathcal{A}$ *with probability about* 2^{-8}.

Property 3 can be explained as follows. The buffer B contains $y_{n+1}, \ldots y_{n+17}$ during the computation of z''_n. Therefore, there exists an integer $1 \le t_n \le 17$, such that

$$z''_n = y_{n+t_n} + (y_n \ggg 16) \bmod 2^{32},$$

We make the heuristic assumption that the high order bits of y_n are uniformly distributed and have verified this assumption with some experiments. With a probability of about 2^{-8} we have $t_n = t_{n+12} = t_{n+17}$. In this case it follows

$$\begin{aligned} S_n &\equiv z''_{n+17} - z''_{n+12} - z''_n \\ &\equiv y_{n+17+t_n} + (y_{n+17} \ggg 16) - y_{n+12+t_n} - (y_{n+12} \ggg 16) - y_{n+t_n} - (y_n \ggg 16) \\ &\equiv (y_{n+17} \ggg 16) - (y_{n+12} \ggg 16) - (y_n \ggg 16) \pmod{2^{32}}. \end{aligned}$$

This value can be evaluated by considering the high order 16 bits and low order 16 bits of y_i separately. We have

$$\begin{aligned} \mathrm{lo}(y_{n+17}) + c2^{16} &= \mathrm{lo}(y_{n+12}) + \mathrm{lo}(y_n) \\ \mathrm{hi}(y_{n+17}) + c'2^{16} &= \mathrm{hi}(y_{n+12}) + \mathrm{hi}(y_n) + c, \end{aligned}$$

where $c, c' \in \{0,1\}^2$ are two carry bits. Hence,

$$\begin{aligned} S_n &\equiv (\mathrm{lo}(y_{n+17}) - \mathrm{lo}(y_{n+12}) - \mathrm{lo}(y_n))2^{16} + \mathrm{hi}(y_{n+17}) - \mathrm{hi}(y_{n+12}) - \mathrm{hi}(y_n) \\ &\equiv -c'2^{16} + c \pmod{2^{32}} \end{aligned}$$

Inserting the four possible values for the pair of carry bits (c, c') results in the set $\mathcal{A}$. Hence, we have given an explanation for the high probability of $S_n \in \mathcal{A}$.

7 An Attack Based on the Small Period of the Lagged Fibonacci Generator

In this section we describe an attack against SSC2 that exploits the rather small period of the lagged Fibonacci generator. The attack requires about 2^{52} words

of known plaintext and the expected time complexity is about $T_0 2^{72.5}$, where T_0 is a constant denoting the work for the innermost loop of our attack. The attack find the internal state of the LFSR. After the internal state of the LFSR is found, we can apply the algorithm described in Section 9 to complete the attack.

The algorithm can be described as follows. Note that we describe the attack in a breadth first version, i.e., here guessing values means choosing all possible values and performing the remaining operations of the algorithm with all possible values in parallel. An implementation of this algorithm would most likely be depth first, but the breadth first version is easier to describe.

1. Let $w = 17(2^{17} - 1)2^{31}$ be the period of the lagged Fibonacci generator. Then compute
$$W_i = z_i \oplus z_{i+w} = z'_i \oplus z'_{i+w}.$$
2. For all i do
   2.1. Let $\mathcal{M} = \{i, i+1, i+2, i+3, i+w, i+w+1, i+w+2, i+w+3\}$.
   2.2. Let $\mathcal{M}' = \{i, \ldots, i+7, i+w, \ldots, i+w+7\}$.
   2.3. For all $m \in \mathcal{M}'$ guess a carry bit g_m for the computation of z'_m.
   2.4. For j from 2 to 16 do
      2.4.1. For all $m \in \mathcal{M}$ guess or extend previous guesses on $\mathrm{hi}(x_m)$, $\mathrm{lo}(x_m)$ to the j least significant bits.
      2.4.2. Use Equation (1) to compute the $j-1$ least significant bits of $\mathrm{hi}(x_m)$ and $\mathrm{lo}(x_m)$ for $m \in \{i+4, \ldots, i+7, i+w+4, \ldots, i+w+7\}$, and the $j-2$ least significant bits of $\mathrm{hi}(x_m)$ and $\mathrm{lo}(x_m)$ for $m \in \{i+8, i+9, i+10, i+w+8, i+w+9, i+w+10\}$.
      2.4.3. Now compute as many bits as possible of
$$v_m = G(x_{m+3}, x_{m+2}, x_{m+1}, x_m, 1, g_m, 1).$$
      That is compute the j least significant bits of $\mathrm{hi}(v_m)$ and $\mathrm{lo}(v_m)$ for $m \in \{i, i+w\}$,
      compute the $j-1$ least significant bits of $\mathrm{hi}(v_m)$ and $\mathrm{lo}(v_m)$ for $m \in \{i+1, i+2, i+3, i+4, i+w+1, i+w+2, i+w+3, i+w+4\}$,
      and compute the $j-2$ least significant bits of $\mathrm{hi}(v_m)$ and $\mathrm{lo}(v_m)$ for $m \in \{i+5, i+6, i+7, i+w+5, i+w+6, i+w+7\}$.
      2.4.4. Now, compute as many bits as possible of $W'_m = v_m \oplus v_{m+w}$ for $i \le m \le i+7$.
      2.4.5. For all $i \le m \le i+7$ compare the known bits W'_m with the corresponding bits of W_m. If any of these bits are different then reject the current guess.
   2.5. For all $i \le m \le i+7$ compute
$$\begin{aligned} \hat{v}_m &= F(x_{m+3}, x_{m+2}, x_{m+1}, x_m) \\ \hat{v}_{m+w} &= F(x_{m+w+3}, x_{m+w+2}, x_{m+w+1}, x_{m+w}) \\ \hat{W}_m &= \hat{z_m} \oplus \hat{z}_{m+w}. \end{aligned}$$
   2.6. For all $i \le m \le i+7$ compare W_m with $\hat{W}_m$. If any of them are different then reject the guess.

First, we have to analyse the consequence of the assumption $d_1 = d_3 = 1$. That is, we are looking for the probability $W_m = W'_m$, where as described earlier W_m is computed using the filter F, but W'_m is computed using the approximation G. We have

$$W'_m \oplus W_m = \Delta_m \oplus \Delta_{m+w}.$$

Hence we can use the distribution of Δ to guess the probability of $W'_m = W_m$. Under the heuristic assumption that Δ_m and Δ_{m+w} are independently distributed we find

$$Prob(W'_m = W_m) = 0.2318.$$

Hence, the number of i we have to check before being successful is $(0.2318)^8 \approx 2^{-16.9}$. Therefore the expected number of key stream words we have to examine before we find such an event is about $2^{17.5}$. (Note this number is slightly larger than $2^{16.9}$ since the events are not independent.)

In Step 2.3 we guess the carry bits g_m for the computations of z'_m for all $m \in \mathcal{M}'$. Hence, we have to guess 16 bits here.

Now we have to divide the analysis into the two cases $j = 2$ and $j > 2$. If $j = 2$, then we guess 16 bits in Step 2.4.1. Note that we do not need to guess the lsb of $\mathrm{lo}(x_i)$ and $\mathrm{lo}(x_{i+w})$. Hence, we guess 30 bits here. We compute 12 bits in the following Steps 2.4.2 through 2.4.4, which gives 12 single bit equations in Step 2.4.5.

If $j > 2$ we have to guess 16 bits in each step, but we also compute 16 new bits in the following equations, which does balance the number of guesses.

Hence, the complexity is

$$2^{17.5} \cdot 2^{16} \cdot 2^{30} \cdot 2^9 \approx 2^{72.5},$$

where the $2^{17.5}$ come from Step 1, 2^{16} from Step 2.3, 2^{30} from Step 2.4.2 with $j = 2$ and the 2^9 is an estimation of the work necessary for every guess in the loop 2.4. Some optimizations that are not described here are still possible. Furthermore, bit slice techniques [Bih97] can be used to speed up the computations.

8 Cryptanalysis of the Lagged Fibonacci Generator

In this section we describe an attack on the lagged Fibonacci generator (endowed with the filter function). The complexity bounds of this attack will be used in the next section and also yield lower bounds on the complexity to find the whole initial state in SSC2.

Assume first that only slightly more than 17 consecutive output words z''_i, $z''_{i+1}, ...$, of the generator are known, and that we want to determine the initial state $y_i, y_{i+1}, ..., y_{i+16}$. After possible reindexing suppose $i = 0$.
Consider the 17 equations (mod 2^{32}), derived by the recurrence relation of the generator,

1. $y_{17} = y_{12} + y_0$
...
5. $y_{21} = y_{16} + y_4$
6. $y_{22} = y_{12} + y_0 + y_5$
...
17. $y_{33} = y_{13} + y_1 + y_6 + y_{11} + y_{16}$

The output words z''_m are computed using the filter function (2). This function is nonlinear, but it becomes linear if we apply mod n cryptanalysis as introduced in [KSW99]. Thereby $n = 2^{32} - 1$ or a prime factor of $2^{32} - 1$ (the prime factors are 3, 5, 17, 257 and $2^{16} + 1$). Recall from [KSW99] that for a 32-bit word x, $x \lll j \equiv 2^j x \bmod 2^{32} - 1$. This implies that using one of the prime factors of $2^{32} - 1$ as a modulus, the 16-bit rotation in equation (2) becomes just multiplication by 1, except for the modulus $2^{16} + 1$, where 16-bit rotation becomes multiplication by -1. The price to pay is an ambiguity in passing from addition mod 2^{32} to addition mod n, depending on whether there was a carry in integer addition mod 2^{32}:

$$(x + y \bmod 2^{32}) \bmod n = \begin{cases} x + y \bmod n \text{ if there was no carry out} \\ x + y - 1 \bmod n \text{ if there was a carry out} \end{cases}$$

To get this relationship note that $2^{32} \equiv 1 \bmod n$ if $n = 2^{32} - 1$ or one of its prime factors.

As a consequence, equation (2) reduces to the two variants

$$z''_m = \varepsilon y_m + y_{m+t_m} \bmod n \text{ or } z''_m = \varepsilon y_m + y_{m+t_m} \bmod n, \tag{3}$$

depending on the carry. Moreover $\varepsilon = 1$ or -1, depending on n. The value of t_m is determined by the 4 most significant bits (msb) of y_{m+17}, which need to be guessed. Considering the 17 recurrence equations, the 4 msb's of $y_{17}, y_{18}, ..., y_{33}$ are guessed, and the equations also hold modulo n if the right sides are subtracted by the appropriate numbers depending on the carry. For equations 1. to 5. the carry needs also to be guessed. However from equation 6. onwards, the carry is mostly determined by the previous choice of the 4 msb's: Consider an arbitrary equation $C = A + B$ of integers mod 2^{32}. If the integer value determined by the 4 msb's of C are smaller than the 4 msb of A (or B), there is a carry, if this value is greater than the 4 msb of A or B, there is no carry. Thus the uncertainty by the carry bits in the 17 equations is about 6 bits. For every choice of the 4 msb's and the carry bits we combine the 17 recurrence equations with equations (3) to get 17 linear equations mod n for y_0, y_1,..., y_{16}. For every prime factor n of $2^{32} - 1$ one solves the system of equations, and the solution mod $2^{32} - 1$ is got by the Chinese remainder theorem. Then the solution found is checked for contradictions with the 4 msb's and the carry's chosen. (In the rare case of a zero value one has to further check whether this value is indeed zero or $2^{32} - 1$.) This procedure is repeated until no contradiction occurs anymore. The remaining candidates are checked by producing one or two further output words z''_m, (which are assumed to be known). Solving systems of linear equations with

17 unknowns is of the order of 2^{12} operations. Hence the total complexity of this analysis is estimated as

$$2^{68} \cdot 2^6 \cdot 2^{17} \cdot 2^{12} = 2^{103}.$$

This analysis can be improved if more output of the generator is assumed to be known. Suppose that $t_m = t_{m+12} = t_{m+17}$. Then S_m takes one of the values in $\mathcal{A}$. Recall that the probability for this event is about 2^{-8}. Let x_1 x_2, x_3 denote the 4-bit integers given by each of the 4 msb's of y_{m+17},..., determining t_m, t_{m+12}, t_{m+17}. Because of the recursion, $x_3 = x_1 + x_2$ or $x_3 = x_1 + x_2 + 1$, if there is a carry from less significant bits. Furthermore the corresponding pointer variables are related by $r_1 = s_1 + 12 \bmod 17$, $s_2 = s_1 + 5 \bmod 17$ and $s_3 = s_1$. Therefore every such pointer variable is expressible in terms of $s_1 := s$. Note that in an attack s can be assumed to be known. Further recall that $1 \leq r, s \leq 17$. Hence arithmetic modulo 17 in the context of the pointers r and s is modified in the sense that a pointer value is added by the value 17 if it becomes 0 or negative. We derive the following equations:

t_m $= ((x_1 + s) \bmod 16)$ - $(s + 12) \bmod 17$
$t_{m+12} = ((x_2 + ((s + 5) \bmod 17)) \bmod 16)$ - s mod 17
$t_{m+17} = ((x_1 + x_2 + s) \bmod 16)$ -$(s + 12) \bmod 17$

or

$t_{m+17} = ((x_1 + x_2 + 1 + s) \bmod 16)$ -$(s + 12) \bmod 17$

if there is a carry. From $t_m = t_{m+17}$ one thus gets $x_2 = 0$ if there is no carry, and $x_2 = 15$ if there is a carry. Moreover, the first two equations and $t_m = t_{m+12}$ imply that a given x_1 and a known s determine the value of x_2. Hence we get a reduction of uncertainty about x_1, x_2 from 16^2 to 2 bits. Further investigation limits the triple (x_1, x_2, x_3) to the values $(0, 0, 0)$, $(15, 15, 15)$, $(1, 0, 1)$, $(15, 0, 15)$ and $(0, 15, 0)$. Experiments show that the frequency of each of these triples is quite different, as different s-values can lead to the same triple. Hence the sequence S_n is not only biased but in case $S_n \in \mathcal{A}$ allows to derive key material.

Suppose now that S_m, S_{m+1}, S_{m+2} and S_{m+3} are all elements of $\mathcal{A}$. This happens with probability about 2^{-32}. Then the uncertainty of the 4 msb's of 8 of the 17 y_n's to be found drops from 16^8 to 2^4 bits. This is a reduction by a factor 2^{28}.

As a consequence, the complexity of our analysis of the filtered lagged Fibonacci generator reduces from 2^{103} to about 2^{75} operations, if the number of known plaintexts is of the order of 2^{32} output words.

9 An Attack Based on the Bias in the Lagged Fibonacci Generator

In this section, we describe briefly an attack against SSC2 that exploits the bias in the lagged Fibonacci generator. This attack requires on average between 2^{45} and 2^{46} known plaintext words and has an expected time complexity of 2^{109}. Because of the large time complexity we only describe the idea behind the attack and omit some details.

Let z_n denote the n-th word of key stream. Then the idea behind the attack is to scan the key stream and hope that for some m the following happens:

- S_m, S_{m+1}, S_{m+2} and S_{m+3} are all elements in the set $\mathcal{A}$.
- For all $n \in \{m, \ldots, m+3, m+12, \ldots, m+15, m+17, \ldots, m+20\}$ the carry bits that occur during the computation of z'_n satisfy the equations $c_1 + c_4 = 1$ and $c_3 + c_5 = 1$.

The probability that $S_m, S_{m+1}, S_{m+2}, S_{m+3} \in \mathcal{A}^4$ is approximately 2^{-32} and the probability that the sum of the carry bits during a computation of z'_n are both 1 is $(2/3)^2$. It follows that all conditions mentioned above are satisfied with a probability of approximately $2^{-32}(2/3)^{24} \approx 2^{-46}$. Hence, we expect to find such an event in 2^{46} words of known key stream.

Thus for all m, let

$$M = \{m, \ldots, m+3, m+12, \ldots, m+15, m+17, \ldots, m+20\}$$

and do the following:

1. Guess the 5 least significant bits of $\mathrm{hi}(x_j)$ and $\mathrm{lo}(x_j)$ for all $m+1 \le j \le m+4$.
2. Guess the carry bit c_2 for all computations of z_n where $n \in M$.
3. For i from 6 to 16 do:
   - 3.1. Guess the i-th least significant bits of $\mathrm{hi}(x_j)$ and $\mathrm{lo}(x_j)$ for all $m+1 \le i \le m+4$.
   - 3.2. Compute the $i-4$ least significant bits of $\mathrm{hi}(x_n)$ and $\mathrm{lo}(x_n)$ for $m \le n \le m+20$.
   - 3.3. Use the computation in figure 4 to compute the $i-4$ least significant bits of $\mathrm{hi}(z'_n)$ and $\mathrm{lo}(z'_n)$ for $n \in M$.
   - 3.4. Use the known key stream z_n to compute z''_n for all $n \in M$.
   - 3.5. For all $m \le n \le m+3$ compute the $i-4$ least significant bits of $\mathrm{hi}(S_n)$ and $\mathrm{lo}(S_n)$. If these bits do not correspond to one of the values in the set $\mathcal{A}$ then reject the guess, otherwise go on with this guess.
4. *At this point we have a guess for all the bits in the linear feedback shift register.* Now, compute the real value of the carry bits. If they don not correspond to the assumptions then reject this guess.
5. Compute more (say 1000) values of S_n and check whether there are more values that are in the set $\mathcal{A}$. If this is not the case then reject the guess.
6. Now, solve for the internal state of the lagged Fibonacci register, as was done in the previous section.

The time complexity of this approach can be computed as follows: We have to go through 2^{46} values for m on average. In step (1) we guess 40 bits. In step (2) we guess 12 carry bits. Hence, up to here we have to guess 98 bits. Then in step (3.1) we guess 8 bits, increasing the complexity to 2^{106}, but in step (3.5) we compute 8 output bits, and hence decrease the complexity at this point to 2^{98} again. Step (4) 2^{72}. So the time requirement for step (5) is insignificant. Hence, the significant part of the complexity comes from step (3). This loop is performed 11 times for about 2^{106} different guesses, giving a total complexity of about 2^{109}.

9.1 Variants

There is a tradeoff possible between the number of known plaintext words and the complexity of the attack:

Instead of assuming that the carry bits that occur during the computation of z'_n satisfy the equations $c_1 + c_4 = 1$ and $c_3 + c_5 = 1$ one may assume this only for a subset of the 24 equations, or one could even do an exhaustive search over all the possible values on the right side of these equations. In the latter case we would need only about 2^{32} (rather than 2^{46}) words or known key stream.

On the average, 8 equations are incorrect, i.e., the right side is either 0 or 2 instead of 1. A rough estimate shows that the complexity for the search with probability 1/2 is not larger than $2\binom{24}{8}2^8 \approx 2^{28}$. Thus with probability about 1/2 the complexity of this variant of the attack is not larger than $2^{32} \cdot 2^{40} \cdot 2^{28} \cdot 2^{12} \cdot 2^8 \cdot 2^3 = 2^{123}$.

10 Frame Key Generation

For synchronization purposes, SSC2 supplies generation of new keys out of the master key for every frame. Each frame is given by a 32-bit number, which is not encrypted. The frame key generation algorithm should satisfy the property that it is difficult to gain information about a frame key from another frame key ([ZCC00]). This property is not satisfied however. The frame key generation is illustrated in [ZCC00] by a pseudo-code, to which we refer. Then one sees that the word $B[1]$ of the key remains constant, i.e. does not depend on the frame number: In line 7 of the code, $B[17 - (i + 8j \bmod 16)] \leftarrow S[1] \oplus B[17 - (i + 8j \bmod 16)]$, the index 1 is never reached.

11 Countermeasures

Certainly, various countermeasures are possible. We think that most importantly the period of the lagged Fibonacci generator should be increased. Even though we do not know how to use the correlation properties of the LFSR alone for an attack, we would still recommend to remove those properties. We have used that the output of the two registers are combined using a simple XOR. Using a nonlinear function instead might further improve the security of the cipher.

12 Conclusion

We have shown some weaknesses of SSC2 and derived two attacks based on those weaknesses. These attacks may not be a threat in the applications, for which SSC2 has been designed. Nonetheless, the results show that the security margin of SSC2 is smaller than previously believed. It is therefore advisable to remove the weaknesses described in this paper. The recent attacks by Hawkes, Quick and Rose [HR01] give even more weight to our opinion.

Acknowledgement. We are thankful to Muxiang Zhan for reviewing a previous version of this paper and pointing out the period in the filter of the lagged Fibonacci generator. We also thank Greg Rose for pointing out an error in a correlation.

References

[Bih97] Eli Biham. A fast new DES implementation in software. In Eli Biham, editor, *Fast Software Encryption '97*, pages 260–272. Springer Verlag, 1997.

[KSW99] J. Kelsey, B. Schneier, and D. Wagner. Mod n cryptanalysis with applications against RC5P and M6. In L. Knudsen, editor, *Fast Software Encryption '99*, volume 1636 of *Lecture Notes in Computer Science*, pages 139–155. Springer Verlag, 1999.

[HR01] P. Hawkes, F. Quick and G. Rose. A practical cryptanalysis of SSC2. In S. Vaudenay and A.M. Youssef, editor, *Selected Areas in Cryptography' 01*, volume 2259 of *Lecture Notes in Computer Science*, pages 27–37, Springer Verlag, 2001.

[SM91] O. Staffelbach and W. Meier. Cryptographic significance of the carry for ciphers based on integer addition. In A.J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology – CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 601–615. Springer Verlag, 1991.

[ZCC00] Muxiang Zhang, Christopher Carroll, and Agnes Chan. The software-oriented stream cipher SSC2. In *Fast Software Encryption '2000 (preproceedings)*, 2000.

Round Security and Super-Pseudorandomness of MISTY Type Structure

Tetsu Iwata, Tomonobu Yoshino, Tomohiro Yuasa, and Kaoru Kurosawa

Department of Communication and Integrated Systems,
Tokyo Institute of Technology
2–12–1 O-okayama, Meguro-ku, Tokyo 152-8552, Japan
{tez,kurosawa}@ss.titech.ac.jp

Abstract. The security of an iterated block cipher heavily depends on its structure as well as each round function. Matsui showed that MISTY type structure is faster and more robust than Feistel structure on linear cryptanalysis and differential cryptanalysis. On the other hand, Luby and Rackoff proved that the four round Feistel structure is super-pseudorandom if each round function f_i is a random function. This paper proves that the five round MISTY type structure is super-pseudorandom. We also characterize its round security.

1 Introduction

The security of an iterated block cipher heavily depends on its structure as well as each round function. There are some well known structures of iterated block ciphers, Feistel structure (for example, DES), MISTY type structure, IDEA type structure and etc. For Feistel structure, Nyberg and Knudsen [7] showed that if each round function is secure against linear cryptanalysis and differential cryptanalysis, then the whole block cipher is immune to both attacks. Matsui showed that MISTY type structure is faster and more robust than Feistel structure on linear cryptanalysis and differential cryptanalysis [4,5].

Pseudorandomness is also an important cryptographic criterion of iterated block ciphers. This approach studies the pseudorandomness of the block cipher by assuming that each round function is ideally random. We say that a block cipher is pseudorandom if it is secure against chosen plaintext attack, where the adversary has access only to the forward direction of the block cipher. It is said to be super-pseudorandom if it is secure under *both* chosen plaintext and chosen ciphertext attacks, where the adversary has access to *both* directions of the block cipher.

The super-pseudorandomness of Feistel structure has been studied extensively so far. Luby and Rackoff proved that the three round Feistel structure is pseudorandom and the four round Feistel structure is super-pseudorandom if each round function f_i is a random function [2]. Patarin gave an alternate proof [8,9]. Lucks showed that the three round Feistel structure is pseudorandom even if the first round function f_1 is an XOR-universal hash function (not

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 233–247, 2002.

necessarily random) [3]. Naor and Reingold showed that the four round Feistel structure is super-pseudorandom even if the first and the last round functions f_1 and f_4 are XOR-universal [6]. Finally, Ramzan and Reyzin showed that the four round Feistel structure is super-pseudorandom even if the adversary has oracle access to the second and the third round functions f_2 and f_3, but not super-pseudorandom if the adversary has oracle access to the first or the last round function, f_1 or f_4 [10].

However, only a little is known about the super-pseudorandomness of MISTY type structure. Sakurai and Zheng showed that the three round MISTY type structure is not pseudorandom, and the four round MISTY type structure is not super-pseudorandom [11]. On the other hand, it is not known if the five round MISTY type structure is super-pseudorandom [11].

This paper characterizes the *super*-pseudorandomness of the five round MISTY type structure. We prove that the five round MISTY type structure is *super*-pseudorandom even if:

1. The first, second and the last round functions, p_1, p_2 and p_5, are XOR-universal permutations. This holds even if the adversary has oracle access to the third and fourth round functions p_3 and p_4.
2. The first and the last round functions, p_1 and p_5, are XOR-universal. This holds even if the adversary has oracle access to the second, third and fourth round functions, p_2, p_3 and p_4.

We also show that it is not super-pseudorandom if the adversary is allowed to have oracle access to the first or the last round function, p_1 or p_5.

Intuitively, our results can be stated as follows. The five round MISTY type structure is super-pseudorandom if: (1) the first and the last rounds have secrecy and only weak randomness, (2) the third and fourth rounds have strong randomness and no secrecy, and (3) the second round has secrecy and only weak randomness, or no secrecy and strong randomness.

To derive our positive results, we use Patarin's approach [8,9] while Ramzan and Reyzin [10] used the approach of Naor and Reingold [6].

Related works: About pseudorandomness (but not super-pseudorandomness) Sugita showed that the four round MISTY type structure is pseudorandom [12], and the five round recursive MISTY type structure is pseudorandom [13].

2 Preliminaries

2.1 Notation

For a bit string $x \in \{0,1\}^{2n}$, we denote the first (left) n bits of x by x_L and the last (right) n bits of x by x_R. If S is a probability space, then $s \xleftarrow{R} S$ denotes the process of picking an element from S according to the underlying probability distribution. (Unless otherwise specified,) The underlying distribution is assumed to be uniform.

Denote by F_n the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$, which consists of $2^{n \cdot 2^n}$ in total. Similarly, denote by P_n the set of all permutations from $\{0,1\}^n$ to $\{0,1\}^n$, which consists of $(2^n)!$ in total. By a finite function (or permutation) family $\mathcal{F}$, we denote a set of functions with common domain and common range. We call a finite function (or permutation) family *keyed* if every function in it can be specified by a key sk. We denote the function given by sk as f_{sk}. We assume that given sk, it is possible to efficiently evaluate f_{sk} at any point (as well as f_{sk}^{-1} in case of a keyed permutation family). For a given keyed function family, a key can be any string from $\{0,1\}^s$, where s is known as "key length." For functions f and g, $g \circ f$ denotes the function $x \mapsto g(f(x))$.

2.2 Super-Pseudorandomness

We are now ready to define a secure block cipher, or what Luby and Rackoff called a super-pseudorandom permutation [2]. The super-pseudorandomness of a keyed permutation family $\mathcal{F}$ over $\{0,1\}^n$ captures its computational indistinguishability from P_n, where the adversary is given access to both directions of the permutation. In other words, it measures security of a block cipher against chosen plaintext and chosen ciphertext attacks.

Our adaptive adversary $\mathcal{M}$ is modeled as a Turing machine that has black-box access to some number k of oracles, each of which computes some specified function. If $(f_1, \ldots, f_k)$ is a k-tuple of functions, then $\mathcal{M}^{f_1,\ldots,f_k}$ denotes a k-oracle adversary who is given black-box access to each of the functions $f_1, \ldots, f_k$. The computational power of $\mathcal{M}$ is unlimited, but the total number of oracle calls is limited to a parameter m.

Definition 2.1. (Advantage, sprp). *Let a block cipher $\mathcal{F}$ be a keyed permutation family over $\{0,1\}^n$ with key length s. Let $\mathcal{M}$ be a 2-oracle adversary. Then we define $\mathcal{M}$'s advantage as*

$$\mathtt{Adv}_{\mathcal{F}}^{\mathsf{sprp}}(\mathcal{M}) \stackrel{\text{def}}{=} |p_f - p_R|$$

where

$$\begin{cases} p_f \stackrel{\text{def}}{=} \Pr(\mathcal{M}^{f_{sk}, f_{sk}^{-1}}(1^n) = 1 \mid sk \xleftarrow{R} \{0,1\}^s) \\ p_R \stackrel{\text{def}}{=} \Pr(\mathcal{M}^{R, R^{-1}}(1^n) = 1 \mid R \xleftarrow{R} P_n) \end{cases}$$

Definition 2.2. (Super-pseudorandom permutation family). *A block cipher $\mathcal{F}$ is super-pseudorandom if $\mathtt{Adv}_{\mathcal{F}}^{\mathsf{sprp}}(\mathcal{M})$ is negligible for any 2-oracle adversary $\mathcal{M}$.*

2.3 MISTY Type Permutation [4,5]

Matsui proposed MISTY [4,5], which is faster and more robust than Feistel structure on linear cryptanalysis and differential cryptanalysis.

Definition 2.3. (The basic MISTY type permutation). *Let $x \in \{0,1\}^{2n}$. For any permutation $p \in P_n$, define the basic MISTY type permutation, $M_p \in P_{2n}$ as $M_p(x) \stackrel{\text{def}}{=} (x_R, p(x_L) \oplus x_R)$. Note that it is a permutation since $M_p^{-1}(x) = (p^{-1}(x_L \oplus x_R), x_L)$.*

Definition 2.4. (The r round MISTY type permutation, ψ). *Let $r \geq 1$ be an integer, $p_1, \ldots, p_r \in P_n$ be permutations. Define the r round MISTY type permutation $\psi(p_1, \ldots, p_r) \in P_{2n}$ as $\psi(p_1, \ldots, p_r) \stackrel{\text{def}}{=} \rho \circ M_{p_r} \circ \cdots \circ M_{p_1}$, where $\rho(x_L, x_R) = (x_R, x_L)$ for $x \in \{0,1\}^{2n}$.*

See Fig. 1 (the four round Feistel permutation) and Fig. 2 (the five round MISTY type permutation) for illustrations. Note that p_i in Fig. 2 is a permutation whereas f_i in Fig. 1 is just a function. For simplicity, the left and right swaps are omitted.

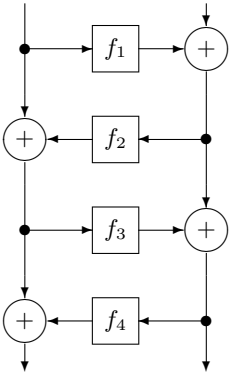


Fig. 1. Feistel permutation

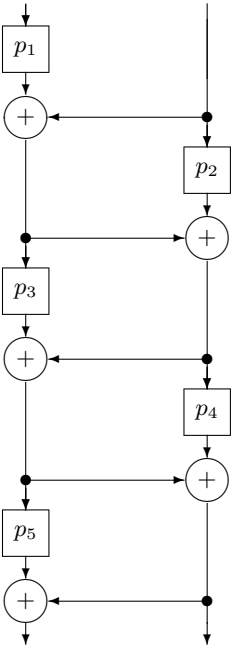


Fig. 2. MISTY type permutation

2.4 Round Security of the Five Round MISTY Type Permutation

The round security model of a block cipher was introduced by Ramzan and Reyzin [10]. In the round security model, the adversary is allowed to have oracle access to some subset K of round functions, and the advantage additionally depends on K.

Definition 2.5. (Round security of the five round MISTY type permutation). *Let p_i be a permutation drawn from a keyed permutation family $\mathcal{P}_i$ over $\{0,1\}^n$ with key length s_i, for $i = 1,\ldots,5$. Let $\psi(p_1,\ldots,p_5)$ be the five round MISTY type permutation, and Ψ be the set of $\psi(p_1,\ldots,p_5)$ with key length $s = s_1 + \cdots + s_5$ (the key sk for $\psi(p_1,\ldots,p_5)$ is simply the concatenation of keys for $p_1,\ldots,p_5$). Fix some subset $\mathcal{K} = \{\Pi^1,\ldots,\Pi^k\}$ of the set $\{\mathcal{P}_1,\mathcal{P}_1^{-1},\ldots,\mathcal{P}_5,\mathcal{P}_5^{-1}\}$, and let $\mathcal{M}$ be a $(k+2)$-oracle adversary. Let $K \stackrel{\text{def}}{=} \{\pi^1_{sk},\ldots,\pi^k_{sk}\}$, where $\pi^i_{sk} \in \Pi^i$ for $1 \le i \le k$. Then we define $\mathcal{M}$'s advantage as*

$$\mathtt{Adv}^{\mathsf{sprp}}_{\Psi;\mathcal{K}}(\mathcal{M}) \stackrel{\text{def}}{=} |p_\psi - p_R|$$

where

$$\begin{cases} p_\psi \stackrel{\text{def}}{=} \Pr(\mathcal{M}^{\psi_{sk},\psi^{-1}_{sk},\pi^1_{sk},\ldots,\pi^k_{sk}}(1^{2n}) = 1 \mid sk \xleftarrow{R} \{0,1\}^s) \\ p_R \stackrel{\text{def}}{=} \Pr(\mathcal{M}^{R,R^{-1},\pi^1_{sk},\ldots,\pi^k_{sk}}(1^{2n}) = 1 \mid R \xleftarrow{R} P_{2n}, sk \xleftarrow{R} \{0,1\}^s) \end{cases}$$

2.5 Uniform ϵ-XOR Universal Permutation

Our definition follows from those given in [1,10,14].

Definition 2.6. *Let H_n be a keyed permutation family over $\{0,1\}^n$. Denote by $\#H_n$ the size of H_n. H_n is* uniform ϵ-XOR universal *provided that the following two conditions are satisfied:*

1. *for any element $x \in \{0,1\}^n$ and any element $y \in \{0,1\}^n$, there exist exactly $\frac{\#H_n}{2^n}$ permutations $h \in H_n$ such that $h(x) = y$.*
2. *for any two distinct elements $x, x' \in \{0,1\}^n$ and any element $y \in \{0,1\}^n$, there exist at most $\epsilon \# H_n$ permutations $h \in H_n$ such that $h(x) \oplus h(x') = y$.*

Let $f_{a,b}(x) \stackrel{\text{def}}{=} a \cdot x + b$ over $\mathrm{GF}(2^n)$, where $a \neq 0$. Then $\{f_{a,b}(x)\}$ is uniform $\frac{1}{2^n-1}$-XOR universal.

We will use the phrase "h is an uniform ϵ-XOR universal permutation" to mean that "h is drawn uniformly from an uniform ϵ-XOR universal permutation family."

3 Round Security of MISTY Type Permutation

3.1 Negative Result

In this section, we show that $\psi(p_1,p_2,p_3,p_4,p_5)$ is not super-pseudorandom if the adversary is allowed to have oracle access to $\{p_1,p_1^{-1}\}$ or $\{p_5^{-1}\}$. This means that, we require secrecy in the first and the last rounds if the cipher is secure.

Theorem 3.1. *Let $p_1,p_2,p_3,p_4,p_5 \in P_n$ be random permutations. Let $\psi = \psi(p_1,p_2,p_3,p_4,p_5)$, and $R \in P_{2n}$ be a random permutation. Suppose that K contains at least one of $\{p_1,p_1^{-1}\}$ or $\{p_5^{-1}\}$. Then there exists an oracle adversary $\mathcal{M}$ such that*

$$\mathtt{Adv}^{\mathsf{sprp}}_{\Psi;\mathcal{K}}(\mathcal{M}) \ge 1 - \frac{2}{2^n} \ .$$

Proof. Let $\mathcal{O} = R$ or ψ. First, suppose that $\mathcal{M}$ has oracle access to $\mathcal{O}$, $\mathcal{O}^{-1}$, p_1 and p_1^{-1}. Consider the following $\mathcal{M}$:

1. Pick $X, A, A' \in \{0,1\}^n$ such that $A \neq A'$ arbitrarily.
2. Ask $(A, A \oplus X)$ to $\mathcal{O}^{-1}$ and obtain (C, D).
3. Ask $(A', A' \oplus X)$ to $\mathcal{O}^{-1}$ and obtain (C', D').
4. Ask C to p_1 and obtain E.
5. Ask C' to p_1 and obtain E'.
6. Ask $D \oplus D' \oplus E$ to p_1^{-1} and obtain F.
7. Ask $D \oplus D' \oplus E'$ to p_1^{-1} and obtain F'.
8. Ask (F', D) to $\mathcal{O}$ and obtain (S, T).
9. Ask (F, D') to $\mathcal{O}$ and obtain (S', T').
10. Output "1" if and only if $S \oplus T = S' \oplus T'$.

If $\mathcal{O} = \psi$, then X is the output of p_5 at step 2 and step 3. Hence the input to p_5 at step 2 is equal to that of step 3. Therefore, from step 4 and step 5, we have

$$p_2(D) \oplus D \oplus E \oplus p_3(D \oplus E) = p_2(D') \oplus D' \oplus E' \oplus p_3(D' \oplus E') \ . \qquad (1)$$

In step 8, the output of p_1 is equal to $D \oplus D' \oplus E'$ from step 7. Therefore, the input to p_5 is equal to $p_2(D) \oplus D' \oplus E' \oplus p_3(D' \oplus E')$ in step 8. Similarly, in step 9, the input to p_5 is equal to $p_2(D') \oplus D \oplus E \oplus p_3(D \oplus E)$. Then from eq.(1), we see that the inputs to p_5 are equal in step 8 and step 9. Hence we have $p_\psi = 1$.

If $\mathcal{O} = R$, we have $p_R \leq \frac{2}{2^n}$.

Next suppose that $\mathcal{M}$ has oracle access to $\mathcal{O}$, $\mathcal{O}^{-1}$ and p_5^{-1}. Consider the following $\mathcal{M}$:

1. Pick $X, B, B' \in \{0,1\}^n$ such that $B \neq B'$ arbitrarily.
2. Ask B to p_5^{-1} and obtain A.
3. Ask B' to p_5^{-1} and obtain A'.
4. Ask $(X, X \oplus B)$ to $\mathcal{O}^{-1}$ and obtain (C, D).
5. Ask $(A \oplus A' \oplus B \oplus B' \oplus X, A \oplus A' \oplus B \oplus X)$ to $\mathcal{O}^{-1}$ and obtain (C', D').
6. Ask (C, D') to $\mathcal{O}$ and obtain (E, F).
7. Ask (C', D) to $\mathcal{O}$ and obtain (E', F').
8. Ask $E \oplus F$ to p_5^{-1} and obtain H.
9. Ask $E' \oplus F'$ to p_5^{-1} and obtain H'.
10. Output "1" if and only if $H \oplus F = H' \oplus F'$.

If $\mathcal{O} = \psi$, then $A \oplus B \oplus X$ is the outputs of p_4 at step 4 and step 5. Then the input to p_4 at step 6 is equal to that of step 7. Hence we have $p_\psi = 1$.

If $\mathcal{O} = R$, we have $p_R \leq \frac{2}{2^n}$. □

3.2 Positive Result 1

Let $h_1, h_2, h_3 \in H_n$ be uniform ϵ-XOR universal permutations and $p \in P_n$ be a random permutation. Let $\psi = \psi(h_1, h_2, p, p, h_3^{-1})$, and $R \in P_{2n}$ be a random permutation. Define $K = \{p, p^{-1}\}$.

Lemma 3.1. *Let m_0 and m_1 be integers. Choose $x^{(i)} \in \{0,1\}^{2n}$ and $y^{(i)} \in \{0,1\}^{2n}$ for $1 \le i \le m_0$ arbitrarily in such a way that $x^{(i)}$ are all distinct and $y^{(i)}$ are all distinct. Choose $X^{(i)} \in \{0,1\}^n$ and $Y^{(i)} \in \{0,1\}^n$ for $1 \le i \le m_1$ arbitrarily in such a way that $X^{(i)}$ are all distinct and $Y^{(i)}$ are all distinct.*

Then the number of (h_1, h_2, p, h_3) such that

$$\left.\begin{array}{l} \psi(x^{(i)}) = y^{(i)} \text{ for } 1 \le \forall i \le m_0, \text{ and} \\ p(X^{(i)}) = Y^{(i)} \text{ for } 1 \le \forall i \le m_1 \end{array}\right\} \tag{2}$$

is at least

$$(\#H_n)^3(2^n - 2m_0 - m_1)! \left(1 - \epsilon \cdot m_0(3m_0 - 2) - \frac{2m_0(m_0 + 2m_1)}{2^n}\right) \ .$$

A proof is given in the next section.

Theorem 3.2. *For any 4-oracle adversary $\mathcal{M}$ that makes at most m queries in total,*

$$\mathtt{Adv}^{\mathsf{sprp}}_{\Psi;K}(\mathcal{M}) \le \epsilon \cdot m(3m-2) + \frac{4m^2}{2^n} \ .$$

Proof. Let $\mathcal{O} = R$ or ψ. The 4-oracle adversary $\mathcal{M}$ has oracle access to $\mathcal{O}$, $\mathcal{O}^{-1}$, p and p^{-1}. Assume that $\mathcal{M}$ makes m_0 queries to $\mathcal{O}$ or $\mathcal{O}^{-1}$, and m_1 queries to p or p^{-1}, where $m = m_0 + m_1$.

Let $q^{(1)}, \ldots, q^{(m_0)} \in \{0,1\}^{2n}$ be bit strings that $\mathcal{M}$ asks to $\mathcal{O}$ or $\mathcal{O}^{-1}$, and let $a^{(1)}, \ldots, a^{(m_0)} \in \{0,1\}^{2n}$ be the answers that $\mathcal{M}$ obtains. Let $Q^{(1)}, \ldots, Q^{(m_1)} \in \{0,1\}^n$ be bit strings that $\mathcal{M}$ asks to p or p^{-1}, and let $A^{(1)}, \ldots, A^{(m_1)} \in \{0,1\}^n$ be the answers that $\mathcal{M}$ obtains.

Let

$$(x^{(i)}, y^{(i)}) = \begin{cases} (q^{(i)}, a^{(i)}) & \text{if } \mathcal{O}(q^{(i)}) = a^{(i)} \\ (a^{(i)}, q^{(i)}) & \text{if } \mathcal{O}^{-1}(q^{(i)}) = a^{(i)} \end{cases} , \tag{3}$$

$$(X^{(i)}, Y^{(i)}) = \begin{cases} (Q^{(i)}, A^{(i)}) & \text{if } p(Q^{(i)}) = A^{(i)} \\ (A^{(i)}, Q^{(i)}) & \text{if } p^{-1}(Q^{(i)}) = A^{(i)} \end{cases} . \tag{4}$$

That is,

$$\mathcal{O}(x^{(i)}) = y^{(i)} \text{ for } 1 \le i \le m_0, \text{ and } p(X^{(i)}) = Y^{(i)} \text{ for } 1 \le i \le m_1 \ .$$

Without loss of generality, we assume that $x^{(i)}$ are all distinct, $y^{(i)}$ are all distinct, $X^{(i)}$ are all distinct and $Y^{(i)}$ are all distinct.

Suppose that $\mathcal{M}$ has obtained $a^{(1)}, \ldots, a^{(i)}$ and $A^{(1)}, \ldots, A^{(j)}$ from the oracles at some point. Then the next behavior of $\mathcal{M}$ is completely determined by $a^{(1)}, \ldots, a^{(i)}$ and $A^{(1)}, \ldots, A^{(j)}$. Therefore, the final output of $\mathcal{M}$ (0 or 1) depends only on $a \stackrel{\text{def}}{=} (a^{(1)}, \ldots, a^{(m_0)})$ and $A \stackrel{\text{def}}{=} (A^{(1)}, \ldots, A^{(m_1)})$. Hence denote by $\mathcal{C}_{\mathcal{M}}(a, A)$ the final output of $\mathcal{M}$.

Let $\mathcal{B} \stackrel{\text{def}}{=} \{(a, A) \mid \mathcal{C}_{\mathcal{M}}(a, A) = 1\}$ and $N \stackrel{\text{def}}{=} \#\mathcal{B}$.

Evaluation of p_R. From the definition of p_R, we have

$$p_R = \Pr_{R,p}(\mathcal{M}^{R,R^{-1},p,p^{-1}}(1^{2n}) = 1) = \frac{\#\{(R,p) \mid \mathcal{M}^{R,R^{-1},p,p^{-1}}(1^{2n}) = 1\}}{(2^{2n})!(2^n)!} \ .$$

We say that (R,p) is compatible with (a,A) if the (R,R^{-1}) oracles answer a and the (p,p^{-1}) oracles answer A. More precisely, (R,p) is compatible with (a,A) if

$$R(x^{(i)}) = y^{(i)} \text{ for } 1 \le i \le m_0 \text{ and } p(X^{(i)}) = Y^{(i)} \text{ for } 1 \le i \le m_1 \ , \qquad (5)$$

where $x^{(i)}, y^{(i)}, X^{(i)}, Y^{(i)}$ are defined by eq.(3) and eq.(4) from (a,A). For each $(a,A) \in \mathcal{B}$, the number of (R,p) which is compatible with (a,A) is exactly $(2^{2n} - m_0)!(2^n - m_1)!$. Therefore, we have

$$\begin{aligned} p_R &= \sum_{(a,A)\in\mathcal{B}} \frac{\#\{(R,p) \mid (R,p) \text{ is compatible with } (a,A)\}}{(2^{2n})!(2^n)!} \\ &= \sum_{(a,A)\in\mathcal{B}} \frac{\#\{(R,p) \mid (R,p) \text{ satisfying (5)}\}}{(2^{2n})!(2^n)!} \\ &= N \cdot \frac{(2^{2n} - m_0)!(2^n - m_1)!}{(2^{2n})!(2^n)!} \ . \end{aligned}$$

Evaluation of p_ψ. From the definition of p_ψ, we have

$$\begin{aligned} p_\psi &= \Pr_{h_1,h_2,p,h_3}(\mathcal{M}^{\psi,\psi^{-1},p,p^{-1}}(1^{2n}) = 1) \\ &= \frac{\#\{(h_1,h_2,p,h_3) \mid \mathcal{M}^{\psi,\psi^{-1},p,p^{-1}}(1^{2n}) = 1\}}{(\#H_n)^3(2^n)!} \ . \end{aligned}$$

Similarly to p_R, we have

$$p_\psi = \sum_{(a,A)\in\mathcal{B}} \frac{\#\{(h_1,h_2,p,h_3) \mid (h_1,h_2,p,h_3) \text{ satisfying (2)}\}}{(\#H_n)^3(2^n)!} \ .$$

Then from Lemma 3.1, we obtain that

$$\begin{aligned} p_\psi &\ge \sum_{(a,A)\in\mathcal{B}} \frac{(\#H_n)^3(2^n - 2m_0 - m_1)!\left(1 - \epsilon \cdot m_0(3m_0 - 2) - \frac{2m_0(m_0+2m_1)}{2^n}\right)}{(\#H_n)^3(2^n)!} \\ &= N\frac{(2^n - 2m_0 - m_1)!}{(2^n)!}\left(1 - \epsilon \cdot m_0(3m_0 - 2) - \frac{2m_0(m_0 + 2m_1)}{2^n}\right) \\ &= p_R\frac{(2^{2n})!(2^n - 2m_0 - m_1)!}{(2^{2n} - m_0)!(2^n - m_1)!}\left(1 - \epsilon \cdot m_0(3m_0 - 2) - \frac{2m_0(m_0 + 2m_1)}{2^n}\right) \ . \end{aligned}$$

Since $\frac{(2^{2n})!(2^n-2m_0-m_1)!}{(2^{2n}-m_0)!(2^n-m_1)!} \geq 1$ (This can be shown easily by an induction on m_0), we have

$$\begin{aligned} p_\psi &\geq p_R \left(1 - \epsilon \cdot m_0(3m_0 - 2) - \frac{2m_0(m_0 + 2m_1)}{2^n}\right) \\ &\geq p_R - \epsilon \cdot m_0(3m_0 - 2) - \frac{2m_0(m_0 + 2m_1)}{2^n} \\ &\geq p_R - \epsilon \cdot m(3m - 2) - \frac{4m^2}{2^n} \ . \end{aligned} \tag{6}$$

Applying the same argument to $1 - p_\psi$ and $1 - p_R$ yields that

$$1 - p_\psi \geq 1 - p_R - \epsilon \cdot m(3m - 2) - \frac{4m^2}{2^n} \ . \tag{7}$$

Finally, (6) and (7) give $|p_\psi - p_R| \leq \epsilon \cdot m(3m - 2) + \frac{4m^2}{2^n}$. □

3.3 Positive Result 2

Let $h_1, h_2 \in H_n$ be uniform ϵ-XOR-universal permutations, $p_1, p_2, p_3 \in P_n$ be random permutations, $\psi = \psi(h_1, p_1, p_2, p_3, h_2^{-1})$, and $R \in P_{2n}$ be a random permutation. Define $K = \{p_1, p_1^{-1}, p_2, p_2^{-1}, p_3, p_3^{-1}\}$.

Lemma 3.2. *Let m_0, m_1, m_2, m_3 be integers such that $m = m_0 + m_1 + m_2 + m_3$. Choose $x^{(i)} \in \{0,1\}^{2n}$ and $y^{(i)} \in \{0,1\}^{2n}$ for $1 \leq i \leq m_0$ arbitrarily in such a way that $x^{(i)}$ are all distinct and $y^{(i)}$ are all distinct. Similarly, for $1 \leq l \leq 3$, choose $X_l^{(i)} \in \{0,1\}^n$ and $Y_l^{(i)} \in \{0,1\}^n$ for $1 \leq i \leq m_l$ arbitrarily in such a way that $X_l^{(i)}$ are all distinct, $Y_l^{(i)}$ are all distinct and*

$$X_1^{(i)} \oplus Y_1^{(i)} \neq X_1^{(j)} \oplus Y_1^{(j)} \text{ for } l = 1 \text{ and } 1 \leq \forall i < \forall j \leq m_1 \ .$$

Then the number of $(h_1, p_1, p_2, p_3, h_2)$ such that

$$\left.\begin{array}{l} \psi(x^{(i)}) = y^{(i)} \text{ for } 1 \leq \forall i \leq m_0, \\ p_1(X_1^{(i)}) = Y_1^{(i)} \text{ for } 1 \leq \forall i \leq m_1, \\ p_2(X_2^{(i)}) = Y_2^{(i)} \text{ for } 1 \leq \forall i \leq m_2, \text{ and} \\ p_3(X_3^{(i)}) = Y_3^{(i)} \text{ for } 1 \leq \forall i \leq m_3 \end{array}\right\} \tag{8}$$

is at least

$$\begin{aligned} &(2^n - m_1)!(2^n - m_2 - m_0)!(2^n - m_3 - m_0)! \\ &\quad (\#H_n)^2 \left(1 - 2\epsilon \cdot m_0(m_0 - 1) - \frac{m_0(2m - 1)}{2^n}\right) \ . \end{aligned}$$

A proof is similar to that of Lemma 3.1.

Theorem 3.3. *For any 8-oracle adversary $\mathcal{M}$ that makes at most m queries in total,*

$$\mathtt{Adv}^{\mathsf{sprp}}_{\Psi;\mathcal{K}}(\mathcal{M}) \le 2\epsilon \cdot m(m-1) + \frac{m(3m-2)}{2^n} \quad .$$

Proof. Let $\mathcal{O} = R$ or ψ. The 8-oracle adversary $\mathcal{M}$ has oracle access to $\mathcal{O}$, $\mathcal{O}^{-1}$, p_1, p_1^{-1}, p_2, p_2^{-1}, p_3 and p_3^{-1}. Assume that $\mathcal{M}$ makes m_0 queries to $\mathcal{O}$ or $\mathcal{O}^{-1}$ and m_l queries to p_l or p_l^{-1} for $1 \le l \le 3$, where $m = m_0 + m_1 + m_2 + m_3$.

Let $q^{(1)}, \ldots, q^{(m_0)} \in \{0,1\}^{2n}$ be bit strings that $\mathcal{M}$ asks to $\mathcal{O}$ or $\mathcal{O}^{-1}$, and let $a^{(1)}, \ldots, a^{(m_0)} \in \{0,1\}^{2n}$ be the answers that $\mathcal{M}$ obtains. For $1 \le l \le 3$, let $Q_l^{(1)}, \ldots, Q_l^{(m_l)} \in \{0,1\}^n$ be bit strings that $\mathcal{M}$ asks to p_l or p_l^{-1}, and let $A_l^{(1)}, \ldots, A_l^{(m_l)} \in \{0,1\}^n$ be the answers that $\mathcal{M}$ obtains.

Let

$$(x^{(i)}, y^{(i)}) = \begin{cases} (q^{(i)}, a^{(i)}) & \text{if } \mathcal{O}(q^{(i)}) = a^{(i)} \\ (a^{(i)}, q^{(i)}) & \text{if } \mathcal{O}^{-1}(q^{(i)}) = a^{(i)} \end{cases} \tag{9}$$

$$(X_l^{(i)}, Y_l^{(i)}) = \begin{cases} (Q_l^{(i)}, A_l^{(i)}) & \text{if } p_l(Q_l^{(i)}) = A_l^{(i)} \\ (A_l^{(i)}, Q_l^{(i)}) & \text{if } p_l^{-1}(Q_l^{(i)}) = A_l^{(i)} \end{cases} \tag{10}$$

for $1 \le l \le 3$. That is, $\mathcal{O}(x^{(i)}) = y^{(i)}$ for $1 \le i \le m_0$ and $p_l(X_l^{(i)}) = Y_l^{(i)}$ for $1 \le l \le 3$ and $1 \le i \le m_l$.

Without loss of generality, we assume that $x^{(i)}$ are all distinct, $y^{(i)}$ are all distinct, $X_l^{(i)}$ are all distinct and $Y_l^{(i)}$ are all distinct, for $1 \le l \le 3$.

Define $a \stackrel{\text{def}}{=} (a^{(1)}, \ldots, a^{(m_0)})$ and $A_l \stackrel{\text{def}}{=} (A_l^{(1)}, \ldots, A_l^{(m_l)})$ for $1 \le l \le 3$. Then, similarly to the proof of Theorem 3.2, we can denote by $\mathcal{C}_{\mathcal{M}}(a, A_1, A_2, A_3)$ the output of $\mathcal{M}$ (0 or 1). Let

$$\begin{cases} \mathcal{B}_1 \stackrel{\text{def}}{=} \{(a, A_1, A_2, A_3) \mid \mathcal{C}_{\mathcal{M}}(a, A_1, A_2, A_3) = 1\}, \\ N \stackrel{\text{def}}{=} \#\mathcal{B}_1, \text{ and} \\ \mathcal{B}_2 \stackrel{\text{def}}{=} \{(a, A_1, A_2, A_3) \mid \mathcal{C}_{\mathcal{M}}(a, A_1, A_2, A_3) = 1 \text{ and} \\ \qquad\qquad 1 \le \forall i < \forall j \le m_1, X_1^{(i)} \oplus Y_1^{(i)} \ne X_1^{(j)} \oplus Y_1^{(j)}\}. \end{cases}$$

Evaluation of p_R. From the definition of p_R, we have

$$\begin{aligned} p_R &= \Pr_{R,p_1,p_2,p_3}(\mathcal{M}^{R,R^{-1},p_1,p_1^{-1},p_2,p_2^{-1},p_3,p_3^{-1}}(1^{2n}) = 1) \\ &= \frac{\#\{(R,p_1,p_2,p_3) \mid \mathcal{M}^{R,R^{-1},p_1,p_1^{-1},p_2,p_2^{-1},p_3,p_3^{-1}}(1^{2n}) = 1\}}{(2^{2n})!\,((2^n)!)^3} \quad . \end{aligned}$$

Since the number of (R, p_1, p_2, p_3) such that

$$\left.\begin{array}{l} R(x^{(i)}) = y^{(i)} \text{ for } 1 \le \forall i \le m_0, \\ p_1(X_1^{(i)}) = Y_1^{(i)} \text{ for } 1 \le \forall i \le m_1, \\ p_2(X_2^{(i)}) = Y_2^{(i)} \text{ for } 1 \le \forall i \le m_2, \text{ and} \\ p_3(X_3^{(i)}) = Y_3^{(i)} \text{ for } 1 \le \forall i \le m_3 \end{array}\right\} \tag{11}$$

is exactly $(2^{2n} - m_0)!(2^n - m_1)!(2^n - m_2)!(2^n - m_3)!$, we have

$$p_R = \sum_{(a,A_1,A_2,A_3)\in\mathcal{B}_1} \frac{\#\{(R,p_1,p_2,p_3) \mid (R,p_1,p_2,p_3) \text{ satisfying (11)}\}}{(2^{2n})!((2^n)!)^3}$$
$$= N \cdot \frac{(2^{2n}-m_0)!(2^n-m_1)!(2^n-m_2)!(2^n-m_3)!}{(2^{2n})!((2^n)!)^3} \quad .$$

Define C be the total number of possible (a, A_1, A_2, A_3). Then

$$C = \frac{(2^{2n})!}{(2^{2n}-m_0)!}\frac{(2^n)!}{(2^n-m_1)!}\frac{(2^n)!}{(2^n-m_2)!}\frac{(2^n)!}{(2^n-m_3)!} \quad .$$

Therefore we have $p_R = \frac{N}{C}$.

Evaluation of p_ψ. From the definition of p_ψ, we have

$$p_\psi = \Pr_{h_1,p_1,p_2,p_3,h_2}(\mathcal{M}^{\psi,\psi^{-1},p_1,p_1^{-1},p_2,p_2^{-1},p_3,p_3^{-1}}(1^{2n}) = 1)$$
$$= \frac{\#\{(h_1,p_1,p_2,p_3,h_2) \mid \mathcal{M}^{\psi,\psi^{-1},p_1,p_1^{-1},p_2,p_2^{-1},p_3,p_3^{-1}}(1^{2n}) = 1\}}{(\#H_n)^2((2^n)!)^3} \quad .$$

Then

$$p_\psi \geq \sum_{(a,A_1,A_2,A_3)\in\mathcal{B}_2} \frac{\#\{(h_1,p_1,p_2,p_3,h_2) \mid (h_1,p_1,p_2,p_3,h_2) \text{ satisfying (8)}\}}{(\#H_n)^2((2^n)!)^3} \quad .$$

Now we want to evaluate $\#\mathcal{B}_2$. Fix any i and j such that $1 \leq i < j \leq m_1$. Then the number of $A_1 = (A_1^{(1)}, \ldots, A_1^{(m_1)})$ which satisfies $X_1^{(i)} \oplus Y_1^{(i)} = X_1^{(j)} \oplus Y_1^{(j)}$ is exactly $\frac{1}{2^n-1}\frac{(2^n)!}{(2^n-m_1)!}$, since we have 2^n choice of $A_1^{(i)}$ which uniquely determines $A_1^{(j)}$ according to the relation $X_1^{(i)} \oplus Y_1^{(i)} = X_1^{(j)} \oplus Y_1^{(j)}$, and other bit strings, $A_1^{(l)}$ where $l \neq i, j$, can be arbitrarily, we have $\frac{(2^n-2)!}{(2^n-m_1)!}$ choice. Since we have $\binom{m_1}{2}$ choice of (i,j), the number of (a, A_1, A_2, A_3) which satisfy

$$1 \leq \exists i < \exists j \leq m_1, X_1^{(i)} \oplus Y_1^{(i)} = X_1^{(j)} \oplus Y_1^{(j)}$$

is at most $\binom{m_1}{2}\frac{C}{2^n-1}$, which is upper bounded by $\frac{m_1(m_1-1)}{2^n}C$. Then it is clear that $\#\mathcal{B}_2 \geq N - \frac{m_1(m_1-1)}{2^n}C$.

Define

$$D \stackrel{\text{def}}{=} \left(1 - 2\epsilon \cdot m_0(m_0-1) - \frac{m_0(2m-1)}{2^n}\right) \quad .$$

Then from Lemma 3.2, we have

$$p_\psi \geq \sum_{(a,A_1,A_2,A_3)\in\mathcal{B}_2} \frac{(2^n-m_1)!(2^n-m_2-m_0)!(2^n-m_3-m_0)!}{((2^n)!)^3}D$$
$$\geq \left(N - \frac{m_1(m_1-1)}{2^n}C\right)\frac{(2^n-m_1)!(2^n-m_2-m_0)!(2^n-m_3-m_0)!}{((2^n)!)^3}D$$
$$= \left(p_R - \frac{m_1(m_1-1)}{2^n}\right)D \cdot C\frac{(2^n-m_1)!(2^n-m_2-m_0)!(2^n-m_3-m_0)!}{((2^n)!)^3} \quad .$$

Since $C\frac{(2^n-m_1)!(2^n-m_2-m_0)!(2^n-m_3-m_0)!}{((2^n)!)^3} = \frac{(2^{2n})!(2^n-m_2-m_0)!(2^n-m_3-m_0)!}{(2^{2n}-m_0)!(2^n-m_2)!(2^n-m_3)!} \geq 1$ (This can be shown easily by an induction on m_0), we have

$$\begin{aligned} p_\psi &\geq \left(p_R - \frac{m_1(m_1-1)}{2^n}\right)\left(1 - 2\epsilon \cdot m_0(m_0-1) - \frac{m_0(2m-1)}{2^n}\right) \\ &\geq p_R - 2\epsilon \cdot m_0(m_0-1) - \frac{m_0(2m-1)}{2^n} - \frac{m_1(m_1-1)}{2^n} \\ &\geq p_R - 2\epsilon \cdot m(m-1) - \frac{m(3m-2)}{2^n} \ . \end{aligned}$$

Then we have

$$|p_\psi - p_R| \leq 2\epsilon \cdot m(m-1) + \frac{m(3m-2)}{2^n}$$

by applying the same argument as was used in Theorem 3.2. □

4 Proof of Lemma 3.1

In ψ, we denote by $I_3 \in \{0,1\}^n$, the input to p in the third round, and denote by $O_3 \in \{0,1\}^n$, the output of it. Similarly, $I_4, O_4 \in \{0,1\}^n$ are the input and output of p in the fourth round, respectively.

Number of h_1. First,

- if $x_L^{(i)} = x_L^{(j)}$, then there exists no h_1 which satisfies
$$h_1(x_L^{(i)}) \oplus x_R^{(i)} = h_1(x_L^{(j)}) \oplus x_R^{(j)} \tag{12}$$
since $x_L^{(i)} = x_L^{(j)}$ implies $x_R^{(i)} \neq x_R^{(j)}$.
- if $x_L^{(i)} \neq x_L^{(j)}$, then the number of h_1 which satisfies (12) is at most $\epsilon \# H_n$ from Definition 2.6.

Therefore, the number of h_1 which satisfies

$$1 \leq \exists i < \exists j \leq m_0, h_1(x_L^{(i)}) \oplus x_R^{(i)} = h_1(x_L^{(j)}) \oplus x_R^{(j)} \tag{13}$$

is at most $\epsilon\binom{m_0}{2}\#H_n$.

Next, the number of h_1 which satisfies

$$h_1(x_L^{(i)}) \oplus x_R^{(i)} = X^{(j)}$$

is exactly $\frac{\#H_n}{2^n}$ from Definition 2.6. Therefore, the number of h_1 which satisfies

$$1 \leq \exists i \leq m_0, 1 \leq \exists j \leq m_1, \ \ h_1(x_L^{(i)}) \oplus x_R^{(i)} = X^{(j)} \tag{14}$$

is at most $\frac{m_0 m_1 \# H_n}{2^n}$.

Then, from (13) and (14), the number of h_1 which satisfies

$$\left.\begin{array}{l} 1 \leq \forall i < \forall j \leq m_0, \ \ h_1(x_L^{(i)}) \oplus x_R^{(i)} \neq h_1(x_L^{(j)}) \oplus x_R^{(j)}, \text{ and} \\ 1 \leq \forall i \leq m_0, 1 \leq \forall j \leq m_1, \ \ h_1(x_L^{(i)}) \oplus x_R^{(i)} \neq X^{(j)} \end{array}\right\} \tag{15}$$

is at least $\#H_n - \epsilon\binom{m_0}{2}\#H_n - \frac{m_0 m_1 \#H_n}{2^n}$. Fix h_1 which satisfies (15) arbitrarily. This implies that $I_3^{(1)}, \ldots, I_3^{(m_0)}$ are fixed in such a way that $I_3^{(1)}, \ldots, I_3^{(m_0)}$ are distinct, and $I_3^{(i)} \neq X^{(j)}$ for $1 \leq \forall i \leq m_0$ and $1 \leq \forall j \leq m_1$.

Number of h_2. Similarly, the number of h_2 which satisfies

$$\left.\begin{array}{l} 1 \leq \forall i < \forall j \leq m_0,\ \ h_2(x_R^{(i)}) \oplus I_3^{(i)} \neq h_2(x_R^{(j)}) \oplus I_3^{(j)}, \\ 1 \leq \forall i \leq m_0, 1 \leq \forall j \leq m_1,\ \ h_2(x_R^{(i)}) \oplus I_3^{(i)} \neq X^{(j)}, \\ 1 \leq \forall i, \forall j \leq m_0,\ \ h_2(x_R^{(i)}) \oplus I_3^{(i)} \neq I_3^{(j)},\ \text{and} \\ 1 \leq \forall i, \forall j \leq m_0,\ \ h_2(x_R^{(i)}) \oplus I_3^{(i)} \neq y_R^{(j)} \end{array}\right\} \tag{16}$$

is at least $\#H_n - \epsilon\binom{m_0}{2}\#H_n - \frac{m_0 m_1 \#H_n}{2^n} - \frac{2m_0^2 \#H_n}{2^n}$. Fix h_2 which satisfies (16) arbitrarily. This implies that $I_4^{(1)}, \ldots, I_4^{(m_0)}$ are fixed in such a way that:

- $I_4^{(1)}, \ldots, I_4^{(m_0)}$ are distinct,
- $I_4^{(i)} \neq X^{(j)}$ for $1 \leq \forall i \leq m_0$ and $1 \leq \forall j \leq m_1$, and
- $I_4^{(i)} \neq I_3^{(j)}$ and $I_4^{(i)} \neq y_R^{(j)}$ for $1 \leq \forall i, \forall j \leq m_0$.

Number of h_3. Similarly, the number of h_3 which satisfies

$$\left.\begin{array}{l} 1 \leq \forall i < \forall j \leq m_0,\ \ h_3(y_L^{(i)} \oplus y_R^{(i)}) \oplus y_R^{(i)} \neq h_3(y_L^{(j)} \oplus y_R^{(j)}) \oplus y_R^{(j)}, \\ 1 \leq \forall i < \forall j \leq m_0,\ \ h_3(y_L^{(i)} \oplus y_R^{(i)}) \oplus I_4^{(i)} \neq h_3(y_L^{(j)} \oplus y_R^{(j)}) \oplus I_4^{(j)}, \\ 1 \leq \forall i \leq m_0, 1 \leq \forall j \leq m_1,\ \ h_3(y_L^{(i)} \oplus y_R^{(i)}) \oplus I_4^{(i)} \neq Y^{(j)}, \\ 1 \leq \forall i \leq m_0, 1 \leq \forall j \leq m_1,\ \ h_3(y_L^{(i)} \oplus y_R^{(i)}) \oplus y_R^{(i)} \neq Y^{(j)},\ \text{and} \\ 1 \leq \forall i, \forall j \leq m_0,\ \ h_3(y_L^{(i)} \oplus y_R^{(i)}) \oplus y_R^{(i)} \neq h_3(y_L^{(j)} \oplus y_R^{(j)}) \oplus I_4^{(j)} \end{array}\right\} \tag{17}$$

is at least $\#H_n - 2\epsilon\binom{m_0}{2}\#H_n - \frac{2m_0 m_1 \#H_n}{2^n} - \epsilon m_0^2 \#H_n$. Fix h_3 which satisfies (17) arbitrarily. This implies that $O_4^{(1)}, \ldots, O_4^{(m_0)}$ and $O_3^{(1)}, \ldots, O_3^{(m_0)}$ are fixed in such a way that:

- $O_4^{(1)}, \ldots, O_4^{(m_0)}$ are distinct,
- $O_3^{(1)}, \ldots, O_3^{(m_0)}$ are distinct,
- $O_3^{(i)} \neq Y^{(j)}$ for $1 \leq \forall i \leq m_0$ and $1 \leq \forall j \leq m_1$,
- $O_4^{(i)} \neq Y^{(j)}$ for $1 \leq \forall i \leq m_0$ and $1 \leq \forall j \leq m_1$, and
- $O_4^{(i)} \neq O_3^{(j)}$ for $1 \leq \forall i, \forall j \leq m_0$.

Number of p. Now h_1, h_2 and h_3 are fixed in such a way that

$$I_3^{(1)}, \ldots, I_3^{(m_0)}, I_4^{(1)}, \ldots, I_4^{(m_0)}, X^{(1)}, \ldots, X^{(m_1)}$$

(which are inputs to p) are all distinct and

$$O_3^{(1)}, \ldots, O_3^{(m_0)}, O_4^{(1)}, \ldots, O_4^{(m_0)}, Y^{(1)}, \ldots, Y^{(m_1)}$$

(which are corresponding outputs of p) are all distinct. In other words, for p, the above $2m_0 + m_1$ input-output pairs are determined. The other $2^n - 2m_0 - m_1$

input-output pairs are undetermined. Therefore we have $(2^n - 2m_0 - m_1)!$ possible choices of p for any such fixed (h_1, h_2, h_3).

Then the number of (h_1, h_2, p, h_3) which satisfy (2) is at least

$$
\begin{aligned}
&(\#H_n)^3(2^n - 2m_0 - m_1)! \left(1 - \epsilon\binom{m_0}{2} - \frac{m_0 m_1}{2^n}\right) \\
&\quad \times \left(1 - \epsilon\binom{m_0}{2} - \frac{m_0 m_1}{2^n} - \frac{2m_0^2}{2^n}\right)\left(1 - 2\epsilon\binom{m_0}{2} - \frac{2m_0 m_1}{2^n} - \epsilon m_0^2\right) \\
&\quad \geq (\#H_n)^3(2^n - 2m_0 - m_1)! \left(1 - \epsilon \cdot m_0(3m_0 - 2) - \frac{2m_0(m_0 + 2m_1)}{2^n}\right)
\end{aligned}
$$

This concludes the proof of the lemma. □

5 Conclusion

In this paper, we proved that:

1. $\psi(p_1, p_2, p_3, p_4, p_5)$ is not super-pseudorandom if the adversary is allowed to have oracle access to $\{p_1, p_1^{-1}\}$ or $\{p_5^{-1}\}$ (Theorem 3.1).
2. $\psi(h_1, h_2, p, p, h_3^{-1})$ is super-pseudorandom even if the adversary has oracle access to p and p^{-1} (Theorem 3.2).
3. $\psi(h_1, p_1, p_2, p_3, h_2^{-1})$ is super-pseudorandom even if the adversary has oracle access to p_1, p_1^{-1}, p_2, p_2^{-1}, p_3 and p_3^{-1} (Theorem 3.3).

The following concrete questions remain to be tackled.

- Is it possible to distinguish $\psi(p_1, p_2, p_3, p_4, p_5)$ from P_{2n} with access to only one of $\{p_1, p_1^{-1}, p_5\}$?
- For example, is $\psi(h_1, p_1, p_1, p_2, h_2^{-1})$ secure when the adversary has oracle access to p_1, p_1^{-1}, p_2 and p_2^{-1} ?

References

1. J.L.Carter and M.N.Wegman. Universal classes of hash functions. *JCSS*, vol. 18, No. 2, pages 143–154, 1979.
2. M.Luby and C.Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, vol. 17, No. 2, pages 373–386, April 1988.
3. S.Lucks. Faster Luby-Rackoff ciphers. *Fast Software Encryption, FSE '96, LNCS* 1039, pages 189–203, Springer-Verlag.
4. M.Matsui. New structure of block ciphers with provable security against differential and linear cryptanalysis. *Fast Software Encryption, FSE '96, LNCS* 1039, pages 206–218, Springer-Verlag.
5. M.Matsui. New block encryption algorithm MISTY. *Fast Software Encryption, FSE '97, LNCS* 1267, pages 54–68, Springer-Verlag.
6. M.Naor and O.Reingold. On the construction of pseudorandom permutations: Luby-Rackoff revised. *J. Cryptology*, vol. 12, No. 1, pages 29–66, Springer-Verlag, 1999.

7. K.Nyberg and L.R.Knudsen. Provable security against a differential attacks. *J. Cryptology*, vol. 8, No. 1, pages 27–37, Springer-Verlag, 1995.
8. J.Patarin. Pseudorandom permutations based on the DES scheme. *Proceedings of Eurocode '90, LNCS* 514, pages 193–204, Springer-Verlag, 1990.
9. J.Patarin. New results of pseudorandom permutation generators based on the DES scheme. *Advances in Cryptology — CRYPTO '91, LNCS* 576, pages 301–312, Springer-Verlag, 1991.
10. Z.Ramzan and L.Reyzin. On the round security of symmetric-key cryptographic primitives. *Advances in Cryptology — CRYPTO 2000, LNCS* 1880, pages 376–393, Springer-Verlag, 2000.
11. K.Sakurai and Y.Zheng. On non-pseudorandomness from block ciphers with provable immunity against linear cryptanalysis. *IEICE Trans. fundamentals*, vol. E80-A, No. 1, pages 19–24, April 1997.
12. M.Sugita. Pseudorandomness of a block cipher MISTY. *Technical report of IEICE*, ISEC 96-9, pages 13–21, 1996.
13. M.Sugita. Pseudorandomness of block ciphers MISTY1. *Technical report of IEICE*, ISEC 97-19, pages 53–64, 1997.
14. M.N.Wegman and J.L.Carter. New hash functions and their use in authentication and set equality. *JCSS*, vol. 22, No. 3, pages 265–279, 1981.

New Results on the Pseudorandomness of Some Blockcipher Constructions

Henri Gilbert and Marine Minier

France Télécom R&D
38-40, rue du Général Leclerc
92794 Issy les Moulineaux Cedex 9
France

Abstract. In this paper, we describe new results on the security, in the Luby-Rackoff paradigm, of two modified Feistel constructions, namely the L-scheme, a construction used at various levels of the MISTY blockcipher which allows to derive a $2n$-bit permutation from several n-bit permutations, and a slightly different construction named the R-scheme. We obtain pseudorandomness and super-pseudorandomness proofs for L-schemes and R-schemes with a sufficient number of rounds, which extend the pseudorandomness and non superpseudorandomness results on the 4-round L-scheme previously established by Sugita [Su96] and Sakurai et al. [Sa97]. In particular, we show that unlike the 3-round L-scheme, the 3-round R-scheme is pseudorandom, and that both the 5-round L scheme and the 5-round R scheme are super pseudorandom (whereas the 4 round versions of both schemes are not super pseudorandom). The security bounds obtained here are close to those established by Luby and Rackoff for the three round version of the original Feistel scheme.

1 Introduction

A key dependent cryptographic function such as a blockcipher can be viewed as a random function associated with a randomly selected key value. It is generally defined using a recursive construction process. Each step of the recursion consists of deriving a random function (or permutation) f from r previously defined random functions (or permutations) $f_1, .., f_r$, and can be represented by a relation of the form $f = \Phi(f_1, .., f_r)$. The most studied example so far is the $f = \Psi(f_1, .., f_r)$ r-round Feistel construction, which allows to derive a $2n$-bit to $2n$-bit random permutation from r n-bit to n-bit functions. But there exist other well known constructions such as for instance Massey and Lai's alternative to the Feistel scheme used in IDEA [La90] and the constructions allowing to deduce a $2n$-bit permutation from several n-bit permutations used in Matsui's MISTY blockcipher [Ma93].

The strongest security requirement one can put on a f random function or permutation representing a key dependent cryptographic function is (informally speaking) that f be undistinguishable with a non negligible success probability from a perfect random function f^* or permutation c^*, even if a probabilistic

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 248–266, 2002.

testing algorithm A of unlimited power is used for that purpose and if the q number of adaptively chosen queries of A to the random instance of f or f^* to be tested is large.

It is generally not possible to prove undistiguishability properties for "real life" cryptologic random function f and large q numbers of queries, because this would require a far too long key length. However, it is often possible to prove or disprove that if a random function f encountered at a given level of a cryptologic function construction is related to random functions encountered at the lower recursion level by a relation of the form $f = \Phi(f_1, .., f_r)$, then if we replace the actual f_1 to f_r random functions of the cipher by independent perfect random functions or permutations f_1^* to f_r^* (or, in a more sophisticated version of the same approach, by f_1' to f_r' functions which are sufficiently undistinguishable from f_1^* to f_r^*), the resulting modified f random function is undistinguishable from a random function (or permutation). This provides a useful method for assessing the soundness of blockcipher constructions. For instance, in the case of a three-round Feistel construction, a well known theorem first proved by Luby and Rackoff [Lu88] provides upper bounds on the $|p - p^*|$ advantage of any A testing algorithm in distinguishing the $f = \Psi(f_1^*, f_2^*, f_3^*)$ $2n$-bit random permutation deduced from three independent ideal random functions f_1^*, f_2^* and f_3^* from a $2n$-bit perfect random permutation c^* with q adaptively chosen queries to the tested instance of f or f^*. This advantage is bounded over by $\frac{q^2}{2^n}$.

The research on pseudorandomness properties of cryptographic constructions initiated Luby and Rackoff's seminal paper [Lu88] has represented a very active research study for the last decade. Just to mention a few examples, Zheng, Matsumoto and Imai and later on Sugita and Sakurai et al. investigated generalised Feistel constructions [Zh89],[Su96],[Su97], Patarin explicited the link between the best advantage of a q-queries distinguisher and the q-ary transition probabilities associated with f and proved undistinguihability bounds for numerous r-round Feistel constructions [Pa91], Maurer showed how to generalise undistinguishability results related to perfect random functions to undistinguishability results related to nearly perfect random functions (e.g. locally random functions)[Ma92], Bellare, Kilian, Rogaway et al. [Be94] investigated the application of similar techniques to modes of operation such as CBC MACs, Aiello and al. proved undistiguihability results on some parallelizable alternatives to the Feistel construction [Ai96] , Vaudenay embedded techniques for deriving undistinguishability bounds into a broader framework he named the decorrelation theory, and applied bounds provided by decorrelation techniques to proving the resistance of actual ciphers, e.g. DFC, against differential and linear cryptanalysis.

In this paper, we describe new results on the security of some blockcipher constructions in the above described paradigm, i.e. we investigate some $f = \Phi(f_1, .., f_k)$ constructions and upper bound the probability of distinguishing f from a perfect random function when Φ is applied to perfect random functions f_i^* or to perfect random permutations c_i^*. We consider alternatives to the Feistel construction allowing to derive a $2n$-bit permutation from several n-bit permutations, namely the so-called L-scheme and R-scheme constructions. The L-type

construction is used for instance at various levels of the construction of Matsui and al. Misty blockcipher [Ma93], as well as in the Kasumi variant of Misty recently adopted as the standard blockcipher for encryption and integrity protection in third generation mobile systems [Ka]. We obtain pseudorandomness and superpseudorandomness proofs for L-scheme and R-scheme constructions with a sufficient number of rounds, which extend the results on the pseudo randomness of the 4-round L-scheme previously established by Sugita [Su96] and Sakurai et al. [Sa97]. In particular, we show that unlike the 3-round L scheme, the 3-round R scheme is pseudorandom, and that both the 5-round L scheme and the 5-round R scheme are super pseudorandom (whereas the 4 round versions of both schemes are not super pseudorandom).

This paper organised as follows: Section 2 introduces basic definitions and useful general results on random functions and techniques for proving that two random functions are undistiguishable. Section 3 describes the R and L schemes. Sections 4 and 5 present our results on the pseudo-randomness and the superpseudorandomness of the L-scheme and the R-scheme respectively, for various numbers of rounds, and Section 6 concludes the paper.

2 Preliminaries

2.1 Notation

Through this paper we are using the following notation: I_n denotes the $\{0,1\}^n$ set. $F_{n,m}$ denotes the ${I_n}^{I_m}$ set of functions from I_n into I_m: thus $|F_{n,m}| = 2^{m.2^n}$. F_n denotes the $F_{n,n}$ set: thus $|F_n| = 2^{n.2^n}$. P_n denotes the set of permutations on I_n: thus $|P_n| = 2^n!$.

2.2 Random Functions

A random function of $F_{n,m}$ is defined as a random variable f of $F_{n,m}$, and can be viewed as a probability distribution $(Pr[f = \varphi])_{\varphi \in F_{n,m}}$ over $F_{n,m}$, or equivalently as a $(f_\omega)_{\omega \in \Omega}$ family of $F_{n,m}$ elements. In particular:

- A n-bit to m-bit key dependent cryptographic function is determined by a randomly selected key value $K \in \mathcal{K}$, and can thus be represented by the random function $f = (f_K)_{K \in \mathcal{K}}$ of $F_{n,m}$.
- A cryptographic construction of the form $f = \Phi(f_1, f_2, .., f_r)$ can be viewed as a random function of $F_{n,m}$ determined by r random functions $f_i \in F_{n_i,m_i}, i = 1..r$

Definition 1. *We define a perfect random function f^* of $F_{n,m}$ as a uniformly drawn element of $F_{n,m}$. In other words, f^* is associated with the uniform probability distribution over $F_{n,m}$. We define a c^* perfect random permutation on I_n as a uniformly drawn element of P_n. In other words, c^* is associated with the uniform probability distribution over P_n.*

Definition 2. *(t-ary transition probabilities associated with f). Given a random function f of $F_{n,m}$, we define the $Pr[x \stackrel{f}{\mapsto} y]$ transition probability associated with a x t-uple of I_n inputs and a y t-uple of I_m outputs as*

$Pr[x \stackrel{f}{\mapsto} y] = Pr[f(x_1) = y_1 \wedge f(x_2) = y_2 \wedge ... \wedge f(x_t) = y_t]$
$= Pr_{\omega \in \Omega}[f_\omega(x_1) = y_1 \wedge f_\omega(x_2) = y_2 \wedge ... \wedge f_\omega(x_t) = y_t]$

In the sequel we will use the following simple properties:

Property 1 *If f^* is a perfect random function $F_{n,m}$ and if $x = (x_1, ..., x_t)$ is a t-uple of pairwise distinct I_n values and if y is any t-uple of I_m values, then $Pr[x \stackrel{f^*}{\mapsto} y] = \frac{1}{|I_m|^t} = 2^{-m.t}$*

Property 2 *Let c^* be a perfect random permutation on I_n. If $x = (x_1, ..., x_t)$ is a t-uple of pairwise distinct I_n values $y = (y_1, ..., y_t)$ is a t-uple of pairwise distinct I_n values then $Pr[x \stackrel{c^*}{\mapsto} y] = (I_n - t)!/|In|! = \frac{(2^n - t)!}{(2^n)!}$*

Property 3 *Let c^* be a perfect random permutation on I_n. If x and x' are two distinct elements of I_n and δ is a given value of I_n then $Pr[c^*(x) \oplus c^*(x') = \delta] \leq \frac{2}{2^n}$.*

Proof: $\Pr[c^*(x) \oplus c^*(x') = 0] = 0$ since $x \neq x'$. If $\delta \neq 0$, $\Pr[c^*(x) \oplus c^*(x') = \delta] = \frac{2^n \cdot 2^{n-2} \cdots 1}{2^n!} = \frac{1}{2^n - 1} \leq \frac{2}{2^n}$. So, $\Pr[c^*(x) \oplus c^*(x') = \delta] \leq \frac{2}{2^n}$.

2.3 Distinguishing Two Random Functions

In proofs of security such as the one presented here, we want to upper bound the probability of any algorithm to distinguish whether a given fixed function φ is an instance of a random function $f = \Phi(f_1^*, f_2^*, .., f_r^*)$ of $F_{n,m}$ or an instance of the perfect random function f^*, using less than q queries to φ.

Let A be any distinguishing algorithm of unlimited power that, when input with a function φ of $F_{n,m}$ (which can be modeled as an " oracle tape " in the probabilistic Turing Machine associated with A) selects a fixed number q of distinct chosen or adaptively chosen input values X_i (the queries), obtains the q corresponding output values $Y_i = f(X_i)$, and based on these results outputs 0 or 1. Denote by p (resp by p^*) the probability for A to answer 1 when fed with a random instance of f (resp of f^*). We want to find upper bounds on the $Adv_A(f, f^*) = |p - p^*|$ advantage of A in distinguishing f from f^* in q queries.

As first noticed by Patarin [Pa91], the best $Adv_A(f, f^*)$ advantage of any A distinguishing algorithm for distinguishing f from f^* is entirely determined by the $Pr[x \stackrel{f}{\mapsto} y]$ q-ary transition probabilities associated with each $X = (X_1, \cdots, X_q)$ q-uple of pairwise distinct I_n values and each $Y = (Y_1, \cdots, Y_q)$ q-uple of I_m values. The following Theorem, which was first proved in [Pa91], and equivalent versions of which can be found in [Va99], is a very useful tool for deriving establishing upper bounds on the $Adv_A(f, f^*)$ based on properties of the $Pr[x \stackrel{f}{\mapsto} y]$ q-ary transition probabilities.

Theorem 1 *Let f be a random function of $F_{n,m}$ and f^* a perfect random function representing a uniformly drawn random element of $F_{n,m}$. Let q be an integer. Denote by $\mathcal{X}$ the ${I_n}^q$ set of all $X = (X_1, \cdots, X_q)$ q-tuples of pairwise distinct elements. If there exists a $\mathcal{Y}$ subset of ${I_m}^q$ and two positive real numbers ϵ_1 and ϵ_2 such that*

1) $|\mathcal{Y}| > (1 - \epsilon_1) \cdot |I_m|^q$ *(i)*

2) $\forall X \in \mathcal{X} \ \forall Y \in \mathcal{Y} Pr[X \stackrel{f}{\mapsto} Y] \geq (1 - \epsilon_2) \cdot \frac{1}{|I_m|^q}$ *(ii)*

then for any A distinguishing using q queries

$$Adv_A(f, f^*) \leq \epsilon_1 + \epsilon_2$$

In order to improve the selfreadability of this paper, a short proof of Theorem 1 is provided in appendix at the end of this paper.

3 Description of the L- and R-Schemes

We now describe two simple variants of the Feistel scheme, that we propose to name L-scheme and R-scheme, following the terminology proposed by Kaneko and al. in their paper on the provable security against differential and linear cryptanalysis of generalised Feistel ciphers [Ka97].

The L-scheme and R-scheme both allow to derive a 2n-bit to 2n-bit permutation from several n-bit to n bit permutations (not only n-bit to n-bit functions as in the Feistel scheme), using only one n-bit to n bit permutation per round.

The 1-round L-scheme is depicted in Figure 1. It transforms a c_1 permutation of I_n into the $\psi_L(c_1)$ permutation of I_{2n} defined by

$$\psi_L(c_1)(x^1, x^0) = (x^0, c_1(x^1) \oplus x^0)$$

The extension to r rounds is straightforward: the r-round L-scheme transforms r I_n permutations c_1 to c_r into the I_{2n} permutation defined by

$$\psi_L(c_1, c_2, ..., c_r) = \psi_L(c_r) \circ \cdots \circ \psi_L(c_1)$$

The L-scheme is used at several levels of the construction of the MISTY and KASUMI ciphers, namely the derivation of the so-called FI and FO functions, and also the upper level of the construction in the case of MISTY2. One remarkable feature of the r-round L-scheme is that two c_i permutations can be processed in paralell.

The 1-round R-scheme is depicted in Figure 1 too. It transforms a $c1$ permutation of I_n into the $\psi_R(c_1)$ permutation of I_{2n} defined by

$$\psi_R(c_1)(x^1, x^0) = (c_1(x^1) \oplus x^0, c_1(x^1))$$

The r-round R-scheme transforms r I_n permutations c_1 to c_r into the I_{2n} permutation defined by $\psi_R(c_1, c_2, ..., c_r) = \psi_R(c_r) \circ \cdots \circ \psi_R(c_1)$.

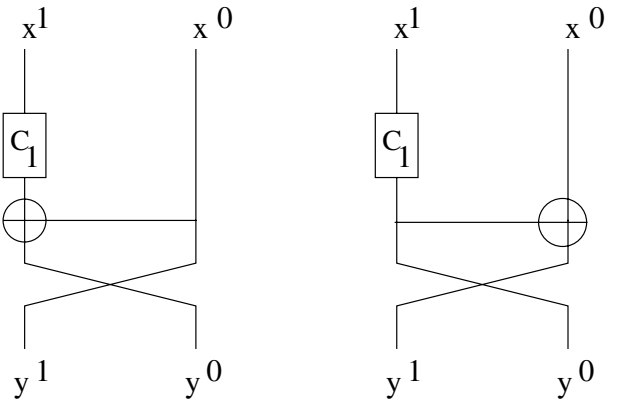


Fig. 1. L-scheme one round at left and R-scheme one round at right

In the sequel, we will several times consider the slightly simplified versions $\psi'_L(c_1, c_2, ..., c_r)$ and $\psi'_R(c_1, c_2, ..., c_r)$ of $\psi_L(c_1, c_2, ..., c_r)$ and $\psi_R(c_1, c_2, ..., c_r)$ obtained by omitting the XOR operation and the exchange of the left and right halves in the final round. We will sometimes analyse such simplified variants, whose pseudorandomness properties are obviously the same as those of the full r-round L or R scheme from which they are derived, instead of the full r-round L or R scheme, in order to simplify some discussions. We can notice that the $\psi'_R(c_1, c_2, ..., c_r)$ and $\psi'_L(c_r^{-1}, c_{r-1}^{-1}, ..., c_1^{-1})$ permutations are inverse of each other. This remark will be useful when it comes to analysing the super pseudorandomness properties of the L and R schemes.

Through the two next Sections, especially in proofs, we are using the following additional notation:

- I is an abreviation for the $(I_n)^q$ set.
- $I^{\neq} = ((I_n)^q))^{\neq}$ denotes the subset of $(I_n)^q$ consisting of all the q-tuples of pairwise distinct I_n values and $I^{=} = (I_n)^q \backslash I^{\neq}$.
- $\mathcal{X}$ denotes the subset of $(I_{2n})^q$ consisting of all $(x_1, \cdots, x_q)$ q-tuples of pairwise distinct I_{2n} values.
- $\mathcal{Y}$ will denote a subset of $(I_{2n})^q$ consisting of $(y_1, \cdots, y_q)$ q-tuples of I_{2n} values. The exact definition of $\mathcal{Y}$ will vary. This $\mathcal{Y}$ will be redefined in each Section where this notation is needed.

4 Analysis of the L-Scheme

In this Section, we compare, for various values of the r number of rounds of an L-scheme, the $f = \psi_L(c_1^*, c_2^*, ..., c_r^*)$ $2n$-bit random permutation deduced from r independent perfect random n-bit permutations $c_1^*, c_2^*, ..., c_r^*$ with a perfect $2n$-bit function f^* or a $2n$-bit perfect permutation c^*.

4.1 Three-Round L-Scheme: $\psi_L(c_1^*, c_2^*, c_3^*)$ Is Not a Pseudo-Random Function

As already noticed by several authors [Zh89], the function $f = \psi_L(c_1^*, c_2^*, c_3^*)$ associated with the three-round L-scheme is not pseudo-random.

Since the omission of the final XOR and the final exchange of the left and right output halves does not affect the pseudorandomness properties of f, we can consider the function $f = \psi'_L(c_1^*, c_2^*, c_3^*)$, instead of $\psi_L(c_1^*, c_2^*, c_3^*)$.

Let us show that 4 chosen input queries suffice to distinguish f from a the perfect random function f^* with a very large probability. Let us consider the encryption, under f, of two distinct $2n$-bit (x^1, x^0) plaintext blocks (a, b) and (a', b) which right halves are equal, and denote by (c, d) and (c', d') the two corresponding (y^1, y^0) ciphertext blocks. We can notice that $d \oplus d'$ is equal to $c_1^*(a) \oplus c_1^*(a')$, and thus independent of b. Therefore if we replace b by any other value b' and do the same computation as before, the new obtained value of $d \oplus d'$ will be left unchanged. This property allows to distinguish f from a perfect random function of I_{2n} with an advantage close to 1.

4.2 Four-Round L-Scheme: $\psi_L(c_1^*, c_2^*, c_3^*, c_4^*)$ Is a Pseudo-Random Function

As already established by Sakurai et al. [Sa97], the four-round version of the L-scheme is indistinguishable from a perfect pseudo-random function. In order for this paper to provide a self contained summary of the properties of the L and R schemes inside the security framework introduced in Section 2, we restate this result as follows.

Theorem 2 *Let n be an integer, c_1^*, c_2^*, c_3^*, c_4^* be four independent random function from I_n to I_n and f^* be the perfect random function on the I_{2n} set. Let $f = \psi_L(f_1^*, f_2^*, f_3^*, f_4^*)$ denote the random permutation associated with the four rounds of L-scheme. For any adaptative distinguisher $\mathcal{A}$ with q queries we have:*

$$Adv_{\mathcal{A}}^{q}(f, f^*) \leq \frac{7}{2} q^2 2^{-n}$$

A short proof for Theorem 2 is provided in appendix at the end of this paper. Since the proof technique is rather similar to the one used in the more detailed proof of Theorem 5 on the pseudorandomness of the 3-round R-scheme, we omitted some details in the proof of Theorem 2.

4.3 Four-Round L-Scheme: $\psi_L(c_1^*, c_2^*, c_3^*, c_4^*)$ Is Not a Super Pseudo-Random Permutation

As already established by Sakurai and al. [Sa97], the 4-round L-scheme does not provide a super pseudo random permutation, i.e. it is possible with a small number of encryption and decryption queries to distinguish $\psi_L(c_1^*, c_2^*, c_3^*, c_4^*)$ from a perfect random permutation.

Instead of providing here a direct proof of this property, let us show that this is a straightforward consequence of the fact (established in the next Section) that the 4-round R-scheme does not provide a super pseudo random function. As a matter of fact, $\psi'_R(c_1, c_2, c_3, c_4)$ and $\psi'_L(c_4^{-1}, c_3^{-1}, c_2^{-1}, c_1^{-1})$ are inverse of each other, as stated in Section 2 and therefore, the distinguisher for $\psi_R(c_1^*, c_2^*, c_3^*, c_4^*)$ can be converted in a distinguisher for $\psi_L(c_1^*, c_2^*, c_3^*, c_4^*)$ with the same number of queries and the same advantage.

4.4 Five-Round L-Scheme: $\psi_L(c_1^*, c_2^*, c_3^*, c_4^*, c_5^*)$ Is a Super Pseudo-Random Permutation

Recall that a super pseudo random distinguisher is an adaptative distinguisher which can call at one and the same time the cipher c and the cipher c^{-1}. The following Theorem shows that the five-round version of the L-scheme provides a super pseudorandom permutation.

Theorem 3 *Let n be an integer, c_1^*, c_2^*, c_3^*, c_4^*, c_5^* be five independent random functions from I_n to I_n and c^* be the perfect random permutation on the I_{2n} set. Let $c = \psi_L(c_1^*, c_2^*, c_3^*, c_4^*, c_5^*)$ denote the random permutation associated with the five round L scheme. For any adaptative super pseudorandom permutation distinguisher $\mathcal{A}$ with q queries, we have:*

$$Adv_{\mathcal{A}}^{q}(c, c^*) \leq \frac{9}{2} \cdot \frac{q^2}{2^n}$$

To prove this theorem, we need to use a variant of Theorem 1 due to Patarin in [Pa91] concerning permutations (that we provide here without proof):

Theorem 4 *Let m be an integer, ϵ be a positive real number, c be a random permutation on the $\{0,1\}^m$ set, c^* be the perfect random permutation on the same set. We denote by $\mathcal{X}$ the subset of $(X_1, \cdots, X_q)$ q-tuples that are pairwise distinct. Let $\mathcal{A}$ be any super pseudo random distinguisher with q queries. If $Pr[X \overset{c}{\mapsto} Y] \geq (1-\epsilon) \cdot \frac{1}{|I_m|^q}$ for all X and Y q-tuples in $\mathcal{X}$ then $Adv_{\mathcal{A}}^{q}(c, c^*) \leq \epsilon + \frac{q(q-1)}{2 \cdot 2^m}$*

Proof of Theorem 3: We will compare the $c = \psi'_L(c_1^*, c_2^*, c_3^*, c_4^*, c_5^*)$ permutation generator of Figure 2 (wich superpseudorandomness properties are exactly the same as for $\psi_L(c_1^*, c_2^*, c_3^*, c_4^*, c_5^*)$) with the perfect random permutation c^* of I_{2n}. For that purpose, let us consider any $X = (x_i^1, x_i^0) \in \mathcal{X}$ q-tuple of pairwise distinct values of I_{2n} and any $Y = (y_i^1, y_i^0)$ q-tuple of pairwise distinct values of I_{2n}. We want to establish lower bound on $\Pr[X \overset{c}{\mapsto} Y]$ and then apply Theorem 4 above. We are using the notation $x^2 = (x_i^2)_{i=1..q}$, $x^3 = (x_i^3)_{i=1..q}$, $x^4 = (x_i^4)_{i=1..q}$ to refer to the q-tuples of I_n intermediate words induced by the q considered f computations, at the locations marked in Figure 2.

$$\Pr[X \overset{c}{\mapsto} Y] = \sum_{x^2, x^3, x^4} \Pr[(c_1^*(x^1) \oplus x^0 = x^2) \wedge (c_2^*(x^0) \oplus x^2 = x^3) \wedge (c_3^*(x^2) \oplus x^3 = x^4)$$

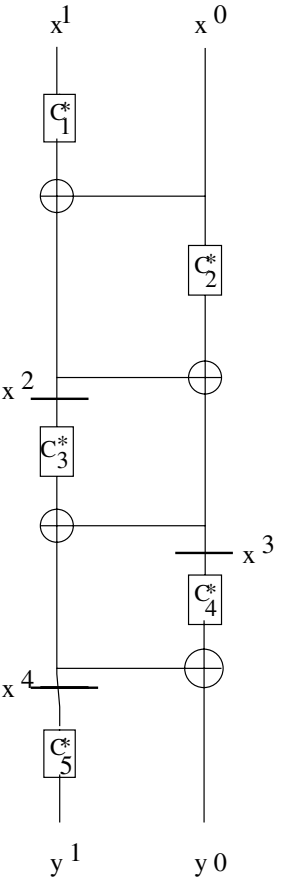


Fig. 2. L-scheme five rounds

$$\begin{aligned} & \wedge(c_4^*(x^3) \oplus x^4 = y^0) \wedge (c_5^*(x^4) = y^1)] \\ \geq \sum_{x^2,x^3 \in I^{\neq}} & \Pr[(c_1^*(x^1) \oplus x^0 = x^2) \wedge (c_2^*(x^0) \oplus x^2 = x^3)] \\ \cdot \sum_{x^4} & \Pr\left[\begin{array}{c}(c_3^*(x^2) \oplus x^3 = x^4) \wedge \\ (c_4^*(x^3) \oplus x^4 = y^0) \wedge (c_5^*(x^4) = y^1)\end{array}\right] \end{aligned} \quad (1)$$

Let us consider any fixed x^2, x^3 q-tuples of $I^{\neq}$. In order to establish a lower bound on the $\sum_{x^4} \Pr[(c_3^*(x^2) \oplus x^3 = x^4) \wedge (c_4^*(x^3) \oplus x^4 = y^0) \wedge (c_5^*(x^4) = y^1)]$ factor in (2), we define the following set Z of x^4 q-tuples:

$$Z = \{x^4 | x^4 \sim y^1 \wedge x^4 \oplus y^0 \in I^{\neq} \wedge x^4 \oplus x^3 \in I^{\neq}\}$$

where $x^4 \sim y^1$ means that $\forall i, j \; x_i^4 = x_j^4$ if and only if $y_i^1 = y_j^1$. Let us denote by $q_1 \leq q$, the number of distinct y_i^1 values. there exist $i_1, \cdots, i_{q_1}$ indexes such that $y_{i_1}^1, \cdots, y_{i_{q_1}}^1$ are pairwise distinct. Each $i_k \in \{i_1, \cdots, i_{q_1}\}$ index determine a class such that for all elements i of this class, $y_i^1 = y_{i_k}^1$. So, $\forall i \in [1, .., q], \exists! i_k \in \{i_1, \cdots, i_{q_1}\} / y_i^1 = y_{i_k}^1, Cl(i) =_{def} i_k$.

There exist $\alpha = \frac{2^n!}{(2^n-q_1)!}$ x^4 values such that $x^4 \sim y^1$ (as a matter of fact such an x^4 is entirely determined by q_1 distinct values). Now:

$$\begin{aligned}|Z| &\geq |\{x^4|x^4 \sim y^1\}| - |\{x^4|x^4 \sim y^1 \wedge x^4 \oplus y^0 \notin I^{\neq}\}| \\ &\quad -|\{x^4|x^4 \sim y^1 \wedge x^4 \oplus x^3 \notin I^{\neq}\}| \\ &\geq |\{x^4|x^4 \sim y^1\}| - \sum_{i \neq j} |\{x^4|x^4 \sim y^1 \wedge x_i^4 \oplus x_j^4 = y_i^0 \oplus y_j^0\}| \\ &\quad - \sum_{i \neq j} |\{x^4|x^4 \sim y^1 \wedge x_i^4 \oplus x_j^4 = x_i^3 \oplus x_j^3\}|\end{aligned}$$

Given $i \neq j$, we can upper bound the size of $S_{ij} = \{x^4|x^4 \sim y^1 \wedge x_i^4 \oplus x_j^4 = y_i^0 \oplus y_j^0\}$ by $\frac{2\alpha}{2^n}$.
As a matter of fact:

- if $y_i^1 = y_j^1$, then $y_i^0 \oplus y_j^0 \neq 0$ (because otherwise the (y_i^1, y_i^0) word would be equal to (y_j^1, y_j^0)), but $x^4 \sim y^1$ implies $x_i^4 = x_j^4$ and thus $x_i^4 \oplus x_j^4$ cannot be equal to $y_i^0 \oplus y_j^0$. So, $|S_{ij}| = 0$
- $y_i^1 \neq y_j^1$, then if $x^4 \in S_{ij}$, $x^4_{Cl(j)}$ is entirely determined by $x^4_{Cl(i)}$ since $x^4_{Cl(j)} = x^4_{Cl(i)} \oplus y_i^0 \oplus y_j^0$. Thus $|S_{ij}|$ contains at most $2^n(2^n-2)\cdots(2^n-q_1) = \frac{\alpha}{2^n-1} \leq \frac{2\alpha}{2^n}$ elements.

Similary, using the fact that $x_i^3 \neq x_j^3$, we can upper bound the size of $\{x^4|x^4 \sim y^1 \wedge x_i^4 \oplus x_j^4 = x_i^3 \oplus x_j^3\}|$ by $\frac{2\alpha}{2^n}$. So, we have:

$$|Z| \geq \alpha \left[1 - \frac{q(q-1)}{2}\frac{2}{2^n} - \frac{q(q-1)}{2}\frac{2}{2^n}\right] \text{ i.e. } |Z| \geq \frac{2^n!}{(2^n-q_1)!}[1 - \frac{2q^2}{2^n}]$$

Now, $\sum_{x^4} \Pr[(c_3^*(x^2) \oplus x^3 = x^4) \wedge (c_4^*(x^3) \oplus x^4 = y^0) \wedge (c_5^*(x^4) = y^1)] \geq \sum_{x^4 \in Z} \Pr[(c_3^*(x^2) \oplus x^3 = x^4) \wedge (c_4^*(x^3) \oplus x^4 = y^0) \wedge (c_5^*(x^4) = y^1)]$. But, for any $x^4 \in Z$, we have $\Pr[c_5^*(x^4) = y^1] = \frac{(2^n-q_1)!}{2^n!} = \frac{1}{\alpha}$ and $\Pr[(c_3^*(x^2) = x^4 \oplus x^3)] = \frac{(2^n-q)!}{2^n!}$ due to Property 2 and the fact that the x_i^2 and the $x_i^4 \oplus x_i^3$ are pairwise distinct. We also have $\Pr[(c_3^*(x^2) = x^4 \oplus x^3)] = \frac{(2^n-q)!}{2^n!}$ for the same reasons, so that:
$\sum_{x^4} \Pr[(c_3^*(x^2) \oplus x^3 = x^4) \wedge (c_4^*(x^3) \oplus x^4 = y^0) \wedge (c_5^*(x^4) = y^1)]$

$$\begin{aligned}&\geq |Z| \cdot \left(\frac{(2^n-q)!}{2^n!}\right)^2 \cdot \frac{1}{\alpha} \\ &\geq \alpha \cdot \left(1 - \frac{2q^2}{2^n}\right)\left(\frac{(2^n-q)!}{2^n!}\right)^2 \cdot \frac{1}{\alpha} \\ &\geq \left(1 - \frac{2q^2}{2^n}\right)\left(\frac{(2^n-q)!}{2^n!}\right)^2\end{aligned}$$

If we now come back to inequality (2), we thus have:

$$\Pr[X \overset{c}{\mapsto} Y] \geq \sum_{x^2,x^3 \in I^{\neq}} \Pr\begin{bmatrix}(c_1^*(x^1) \oplus x^0 = x^2)\\ \wedge \\ (c_2^*(x^0) \oplus x^2 = x^3)\end{bmatrix} \cdot \left(1 - \frac{2q^2}{2^n}\right)\left(\frac{(2^n-q)!}{2^n!}\right)^2 \quad \text{(i)}$$

Let us now establish a lower bound on

$$\begin{aligned} B &= \sum_{x^2,x^3 \in I^{\neq}} \Pr[(c_1^*(x^1) \oplus x^0 = x^2) \wedge (c_2^*(x^0) \oplus x^2 = x^3)] \\ &= \Pr[x^2 \in I^{\neq} \wedge x^3 \in I^{\neq}] \\ &\quad \cdot \Pr[(c_1^*(x^1) \oplus x^0) \in I^{\neq} \wedge (c_2^*(x^0) \oplus x^2) \in I^{\neq} | x^2 \in I^{\neq}] \end{aligned}$$

But $\Pr[(c_1^*(x^1) \oplus x^0) \in I^{\neq}] \geq 1 - \sum_{i \neq j} \Pr[c_1^*(x_i^1) \oplus c_1^*(x_j^1) = x_i^0 \oplus x_j^0]$ and it is easy to establish (using the fact that $(x_i^1, x_i^0) \neq (x_j^1, x_j^0)$ and Property 3), that for any two distinct indexes i and j, $\Pr[c_1^*(x_i^1) \oplus c_1^*(x_j^1) = x_i^0 \oplus x_j^0] \leq \frac{1}{2^n - 1} \leq \frac{2}{2^n}$ Thus $\Pr[(c_1^*(x^1) \oplus x^0) \in I^{\neq}] \geq 1 - \frac{q(q-1)}{2} \cdot \frac{2}{2^n} \geq 1 - \frac{q^2}{2^n}$. for similar reasons, $\Pr[(c_2^*(x^0) \oplus x^2) \in I^{\neq} | x^2 \in I^{\neq}] \geq 1 - \frac{q^2}{2^n}$. Thus $B \geq \left(1 - \frac{q^2}{2^n}\right)^2 \geq 1 - \frac{2q^2}{2^n}$ (ii) . Now, by combinig (i) and (ii), we obtain:

$$\Pr[X \overset{c}{\mapsto} Y] \geq \left(1 - \frac{2q^2}{2^n}\right)^2 \left(\frac{(2^n-q)!}{2^n!}\right)^2$$

Now, $\left(\frac{(2^n-q)!}{2^n!}\right)^2 \geq \frac{1}{2^{2nq}}$ and $\left(1 - \frac{2q^2}{2^n}\right)^2 \geq 1 - \frac{4q^2}{2^n}$, so that

$$\Pr[X \overset{c}{\mapsto} Y] \geq \left(1 - \frac{4q^2}{2^n}\right) \cdot \frac{1}{2^{2nq}}$$

We can now apply Theorem 4 with $\epsilon = \frac{4q^2}{2^n}$ and we obtain:

$$\begin{aligned} \mathrm{Adv}_{\mathcal{A}}^q(c, c^*) &\leq \tfrac{4q^2}{2^n} + \tfrac{q(q-1)}{2_2 2^{2n}} \\ \mathrm{Adv}_{\mathcal{A}}^q(c, c^*) &\leq \tfrac{9}{2} \cdot \tfrac{q^2}{2^n} \end{aligned}$$

□

5 Analysis of the R-Scheme

In this Section, we compare, for various values of the r number of rounds of an R-scheme, the $f = \psi_R(c_1^*, c_2^*, ..., c_r^*)$ $2n$-bit random permutation deduced from r independent perfect random n-bit permutations $c_1^*, c_2^*, ..., c_r^*$ with a perfect $2n$-bit function f^*.

5.1 Three-Round R-Scheme

We first establish the following theorem for a 3-round version of the R-scheme.

Theorem 5 *Let n be an integer, c_1^*, c_2^*, c_3^* be three independent perfect random permutation from I_n to I_n and f^* be the perfect random function on the I_{2n} set. Let $f = \psi_R(c_1^*, c_2^*, c_3^*)$ denote the random permutation associated with the 3-rounds R-scheme. For any adaptive distinguisher $\mathcal{A}$ with q queries, we have:*

$$Adv_{\mathcal{A}}^{q}(f, f^*) \leq 3q^2 2^{-n}$$

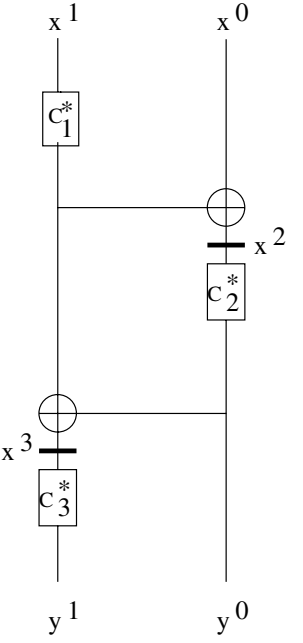


Fig. 3. R-scheme three rounds

Proof: We will compare the $f = \psi'_R(c_1^*, c_2^*, c_3^*)$ permutation generator of Figure 3 (which pseudorandomness properties are exactly the same as for $\psi_R(c_1^*, c_2^*, c_3^*)$) with the perfect random function f^*. Let us first introduce some notation. We consider a $X = (X_i)_{i\in[1..q]} = (x_i^1, x_i^0)$ q-tuple of $2n$-bit f input words. We denote the corresponding q-tuple of f output words by $Y = (Y_i)_{i\in[1..q]} = (y_i^1, y_i^0)_{i\in[1..q]}$. For each $(y_i^1, y_i^0) = \psi'_R(c_1^*, c_2^*, c_3^*)(x_i^1, x_i^0)$ computation, we denote by x_i^2 and x_i^3 the intermediate values which locations are marked in Figure 3. More explicitly:

$$x_i^2 = c_1^*(x_i^1) \oplus x_i^0$$
$$x_i^3 = c_1^*(x_i^1) \oplus c_2^*(x_i^2) = y_i^0 \oplus c_1^*(x_i^1)$$

Finally, we denote the $(x_i^0)_{i\in[1..q]}$, $(x_i^1)_{i\in[1..q]}$, $(x_i^2)_{i\in[1..q]}$, $(x_i^3)_{i\in[1..q]}$, $(y_i^0)_{i\in[1..q]}$ and $(y_i^1)_{i\in[1..q]}$ q-tuples of n-bit words by x^0, x^1, x^2, x^3, y^0, y^1 respectively.

We now define $\mathcal{X}$ as the set of X q-tuples of pairwise distinct I_{2n} words (i.e. such that for any distinct i,j numbers in $[1..q]$, $x_i^1 \neq x_j^1$ or $x_i^0 \neq x_j^0$), and define $\mathcal{Y}$ as the set of those Y q-tuples of I_{2n} words such that the corresponding y^1 and y^0 q-tuples both consist of pairwise distinct I_n words: $\mathcal{Y} = \{(Y_1, \cdots, Y_q) \in (I_{2n})^q \ / \ y^1 \in I^{\neq},\ y^0 \in I^{\neq}\}$.

We want to establish a lower bound on the size of $\mathcal{Y}$ and the $\Pr[X \overset{f}{\mapsto} Y]$ transition probability associated with any X q-tuple in $\mathcal{X}$ and any Y q-tuple in $\mathcal{Y}$ and show that there exists ϵ_1 and ϵ_2 real numbers satisfying conditions of Theorem 1.

Let us first establish a lower bound on $|\mathcal{Y}|$. We have:

$$\begin{aligned}
|\mathcal{Y}| &= |I^{2n}|^q \cdot (1 - \Pr[y^1 \notin I^{\neq} \vee y^0 \notin I^{\neq}]) \\
&\geq |I^{2n}|^q (1 - \sum_{i,j\in[1..q], i\neq j} \Pr[y_i^1 = y_j^1] - \sum_{i,j\in[1..q], i\neq j} \Pr[y_i^0 = y_j^0]) \\
&\geq |I^{2n}|^q (1 - \frac{q(q-1)}{2} \cdot 2^{-n} - \frac{q(q-1)}{2} \cdot 2^{-n}) \\
&\geq |I^{2n}|^q (1 - \frac{q(q-1)}{2^n})
\end{aligned}$$

So, we can take $\epsilon_1 = \frac{q(q-1)}{2^n}$.

Now, given any X q-tuple of $\mathcal{X}$ and any Y q-tuple of $\mathcal{Y}$ let us establish a lower bound on $\Pr[X \overset{f}{\mapsto} Y]$.

$$\begin{aligned}
\Pr[X \overset{f}{\mapsto} Y] &= \textstyle\sum_{x^2,x^3\in I} \Pr[(c_1^*(x^1) \oplus x^0 = x^2) \wedge (c_1^*(x^1) \oplus y^0 = x^3) \\
&\qquad \wedge (c_2^*(x^2) = y^0) \wedge (c_3^*(x^3) = y^1)] \\
&\geq \textstyle\sum_{x^2,x^3\in I^{\neq}} \Pr[(c_1^*(x^1) \oplus x^0 = x^2) \wedge (c_1^*(x^1) \oplus y^0 = x^3)] \\
&\qquad \cdot \Pr[(c_2^*(x^2) = y^0) \wedge (c_3^*(x^3) = y^1)] \qquad (2)
\end{aligned}$$

First, for any x^2 q-tuple of $I^{\neq}$ and any x^3 q-tuple of $I^{\neq}$, let us compute $\Pr[(c_2^*(x^2) = y^0) \wedge (c_3^*(x^3) = y^1)] = \Pr[(c_2^*(x^2) = y^0)] \cdot \Pr[(c_3^*(x^3) = y^1)]$. Since x^2 and y^0 both belong to $I^{\neq}$, we can can apply Property 2 of Section 2 concerning random permutations, so that $\Pr[(c_2^*(x^2) = y^0)] = \frac{(2^n-q)!}{2^n!}$. For the same reason, we also have $\Pr[(c_3^*(x^3) = y^1)] = \frac{(2^n-q)!}{2^n!}$.

So, we have $\Pr[(c_2^*(x^2) = y^0) \wedge (c_3^*(x^3) = y^1)] = \left(\frac{(2^n-q)!}{2^n!}\right)^2$.

Now, $\left(\frac{(2^n-q)!}{2^n!}\right)^2 \geq \frac{1}{2^{2nq}}$.

Therefore, inequality (2) implies:

$$\Pr[X \overset{f}{\mapsto} Y] \geq \sum_{x^2,x^3\in I^{\neq}} \frac{1}{2^{2nq}} \cdot \Pr[(c_1^*(x^1) \oplus x^0 = x^2) \wedge (c_1^*(x^1) \oplus y^0 = x^3)] \quad (i)$$

Let us now estimate $B = \sum_{x^2,x^3 \in I^{\neq}} \Pr[(c_1^*(x^1) \oplus x^0 = x^2) \wedge (c_1^*(x^1) \oplus y^0 = x^3)]$

We have:

$$\begin{aligned} B &= \Pr[(c_1^*(x^1) \oplus x^0 \in I^{\neq} \wedge (c_1^*(x^1) \oplus y^0 \in I^{\neq}] \\ &= 1 - \Pr[(c_1^*(x^1) \oplus x^0 \notin I^{\neq} \vee (c_1^*(x^1) \oplus y^0 \notin I^{\neq}] \\ &\geq 1 - \sum_{i,j,i\neq j} Pr[c_1^*(x_i^1) \oplus x_i^0 = c_1^*(x_j^1) \oplus x_j^0] \\ &\quad - \sum_{i,j,i\neq j} Pr[c_1^*(x_i^1) \oplus y_i^0 = c_1^*(x_j^1) \oplus y_j^0] \end{aligned}$$

Let us evaluate $\Pr[c_1^*(x_i^1) \oplus x_i^0 = c_1^*(x_j^1) \oplus x_j^0]$ and $\Pr[c_1^*(x_i^1) \oplus c_1^*(x_j^1) = x_i^0 \oplus x_j^0]$. Due to Property 3 of Section 2, if $x_i^1 \neq x_j^1$ given any fixed difference δ (here equal to $x_i^0 \oplus x_j^0$), $\Pr[c_1^*(x_i^1) \oplus c_1^*(x_j^1) = \delta] \leq \frac{2}{2^n}$. On the other hand, if $x_i^1 = x_j^1$, then $x_i^0 \neq x_j^0$, so that $\Pr[c_1^*(x_i^1) \oplus x_i^0 = c_1^*(x_j^1) \oplus x_j^0]$. In all cases, $\Pr[c_1^*(x_i^1) \oplus x_i^0 = c_1^*(x_j^1) \oplus x_j^0] \leq \frac{2}{2^n}$. Applying this property to the $\frac{q(q-1)}{2}$ $(i,j), i \neq j$ pairs of $[1..q]$ indexes, we obtain $\sum_{i,j} Pr[c_1^*(x_i^1) \oplus x_i^0 = c_1^*(x_j^1) \oplus x_j^0] \leq \frac{q(q-1)}{2^n}$. For the same reasons, $\sum_{i,j,i\neq j} Pr[c_1^*(x_i^1) \oplus y_i^0 = c_1^*(x_j^1) \oplus y_j^0] \leq \frac{q(q-1)}{2^n}$. Thus:

$$B \geq 1 - \frac{2q(q-1)}{2^n} \quad (ii)$$

By using inequalities (i) and (ii), we obtain:

$$\Pr[X \overset{f}{\mapsto} Y] \geq \left(1 - \frac{2q(q-1)}{2^n}\right) \cdot \frac{1}{2^{2nq}}$$

We can notice that $Pr[X \overset{f^*}{\mapsto} Y] = \frac{1}{2^{2nq}}$. So we can apply Theorem 1 with $\epsilon_1 = \frac{q(q-1)}{|I^n|}$ and $\epsilon_2 = \frac{2q(q-1)}{|I^n|}$. We obtain:

$$\begin{gathered} \mathrm{Adv}_{\mathcal{A}}^q(f, f^*) \leq \frac{3q(q-1)}{2^n} \\ \mathrm{Adv}_{\mathcal{A}}^q(f, f^*) \leq \frac{3q^2}{2^n} \end{gathered}$$

□

5.2 Four-Round R-Scheme: $\psi_R(c_1^*, c_2^*, c_3^*, c_4^*)$ Is Not a Super Pseudo-Random Permutation

We consider the four-round permutation generator f deduced from $\psi_R(c_1^*, c_2^*, c_3^*, c_4^*)$ by omitting the final XOR (this does obviously not matter for the super pseudo randomness issue considered here). The random function f can be represented by extending the 3-round function of Figure 3 above by one round.

Let us show that 2 chosen encryption and two chosen decryption queries suffice to distinguish f from a the perfect random permutation $c*$ with a very large probability. Let us consider the encryption, under the function f, of two

distinct $2n$-bit (x^1, x^0) plaintext blocks (a, b) and (a, b') which left halves are equal, and denote by (c, d) and (c', d') the two obtained (y^1, y^0) ciphertext blocks. It is easy to check that if we swap the left halves of the two obtained ciphertexts, thus obtaining two modified ciphertexts (c', d) and (c, d') and if we decrypt (c', d) and (c, d') under f^{-1}, we obtain two plantext values α, β and α', β' which left halves are equal: $\alpha = \alpha'$. This would be extremely unlikely to happen if f was replaced by a perfect random permutation $c*$.

The above test allows to distinguish f from a perfect random permutation of I_{2n} with a probability close to 1.

5.3 Five-Round R-Scheme: $\psi_R(c_1^*, c_2^*, c_3^*, c_4^*, c_5^*)$ Is a Super Pseudo-Random Permutation

The following theorem is a direct consequence of Theorem 3 due to the fact that $\psi'_R(c_1, c_2, c_3, c_4, c_5)$ and $\psi'_L(c_1^{-1}, c_2^{-1}, c_3^{-1}, c_4^{-1}, c_5^{-1})$ are inverse of each other, every distinguisher for $\psi_R(c_1^*, c_2^*, c_3^*, c_4^*, c_5^*)$ can be converted into a distinguisher for $\psi_L(c_1^*, c_2^*, c_3^*, c_4^*, c_5^*)$ with the same number of encryption and decryption queries. Therefore, Theorem 3 results in the following analogue theorem for the 5-round R-scheme.

Theorem 6 *Let n be an integer, c_1^*, c_2^*, c_3^*, c_4^*, c_5^* be five independent random functions from I_n to I_n and c^* be the perfect random permutation on the I_{2n} set. Let $c = \psi_R(c_1^*, c_2^*, c_3^*, c_4^*, c_5^*)$ denote the random permutation associated with the five round R-scheme. For any adaptive super pseudorandom permutation distinguisher $\mathcal{A}$ with q queries, we have:*

$$Adv_{\mathcal{A}}^{q}(c, c^*) \leq \frac{9}{2} \cdot \frac{q^2}{2^n}$$

6 Conclusion

As a consequence of previous results, the security properties of the L-scheme and the R-scheme are distinct when it comes to chosen plaintext attacks, but equivalent when it comes to chosen plaintext or ciphertext attacks. As a matter of fact, the minimal number of rounds required in order of the R-scheme to be undistinguishable from a pseudorandom function with adaptively chosen encryption queries is less than for the L-scheme (3 rounds instead of 4), whereas the minimal numbers of rounds required by the R-scheme and the L-scheme in order to be undistinguishable from a pseudorandom permutation with adaptively chosen encryption or decryption queries are equal (5 rounds for both schemes).

A Appendix

A.1 A Short Proof of Theorem 1

Let us restrict ourselves to the case of any fixed deterministic algorithm A which uses q adaptively chosen queries (the generalisation to the case of a probabilistic algorithm is easy).

A has the property that if the q-uple of outputs encountered during an A computation is $Y = (Y_1, \cdots, Y_q)$, the value of the $X = (X_1, \cdots, X_q)$ q-uple n is of query inputs encountered during this computation is entirely determined (this is easy to prove by induction: the initial query input X_1 is fixed ; if for a given A computation the first query output is Y_1, then X_2 is determined, etc.). We denote by $X(Y)$ the single q-uple of query inputs corresponding to any possible Y q-uple of query outputs, and we denote by S_A the subset of those $Y \in {I_m}^q$ values such that if the $X(Y)$ and Y q-uples query inputs and outputs are encountered in a A computation, then A outputs a 1 answer.

The p and p^* probabilities can be expressed using S_A as

$$p = \sum_{Y \in S_A} Pr[X(Y) \stackrel{f}{\mapsto} Y] \text{ and } p^* = \sum_{Y \in S_A} Pr[X(Y) \stackrel{f^*}{\mapsto} Y]$$

We can now lower bound p using the following inequalities:

$p \geq \sum_{Y \in S_A \cap \mathcal{Y}} (1 - \epsilon_2) \cdot Pr[X(Y) \stackrel{f^*}{\mapsto} Y]$ (due to inequality (ii))

$\geq \sum_{Y \in S_A} (1 - \epsilon_2) Pr[X(Y) \stackrel{f^*}{\mapsto} Y] - \sum_{Y \in {I_m}^{q-} mathcal Y} (1 - \epsilon_2) \cdot Pr[X(Y) \stackrel{f^*}{\mapsto} Y]$

But

$$\sum_{Y \in S_A} (1 - \epsilon_2) \cdot Pr[X(Y) \stackrel{f^*}{\mapsto} Y] = (1 - \epsilon_2) p^*$$

and

$$\sum_{Y \in {I_m}^q - \mathcal{Y}} (1 - \epsilon_2) \cdot Pr[X(Y) \stackrel{f^*}{\mapsto} Y] = (1 - \epsilon_2) \cdot \frac{|I_m|^q - |\mathcal{Y}|}{|I_m|} \leq (1 - \epsilon_2) \cdot \epsilon_1 \text{ (due to inequality (i)).}$$

Therefore $p \geq (1 - \epsilon_2)(p^* - \epsilon_1) = p^* - \epsilon_1 - \epsilon_2 \cdot p^* + \epsilon_1 \cdot \epsilon_2$

thus finally (using $p^* \leq 1$ and $\epsilon_1 \cdot \epsilon_2 \geq 0$)

$p \geq p* - \epsilon_1 - \epsilon_2$ (a)

If we now consider the A' distinguisher which outputs are the inverse of those of A (i.e. A' answers 0 iff A answers 1), we obtain an inequality involving this time $1 - p$ and $1 - p*$:

$(1 - p) \geq (1 - p*) - \epsilon_1 - \epsilon_2$ (b)

Combining inequalities (a) and (b), we obtain $|p - p*| \leq \epsilon_1 + \epsilon_2$ QED.

A.2 A Proof Sketch for Theorem 2

We will compare the $f = \psi'_L(c_1^*, c_2^*, c_3^*, c_4^*)$ permutation generator of Figure 4 (which pseudorandomness properties are exactly the same as for $\psi_L(c_1^*, c_2^*, c_3^*, c_4^*)$) with the perfect random function f^*. This proof is near to the proof of Section 5.1. That's why we do not detail some computations that are the same that in Section 5.1.

Let us first introduce some notation. We consider a $X = (X_i)_{i \in [1..q]} = (x_i^1, x_i^0)$ q-tuple of 2n-bit f input words. We denote the corresponding q-tuple of f output words by $Y = (Y_i)_{i \in [1..q]} = (y_i^1, y_i^0)_{i \in [1..q]}$. For each $(y_i^1, y_i^0) = \psi'_L(c_1^*, c_2^*, c_3^*, c_4^*)(x_i^1, x_i^0)$ computation, we denote by x_i^2 and x_i^3 the intermediate values which locations are marked in Figure 4. More explicitly:

$$x_i^2 = c_1^*(x_i^1) \oplus x_i^0$$
$$x_i^3 = c_2^*(x_i^0) \oplus x_i^2$$

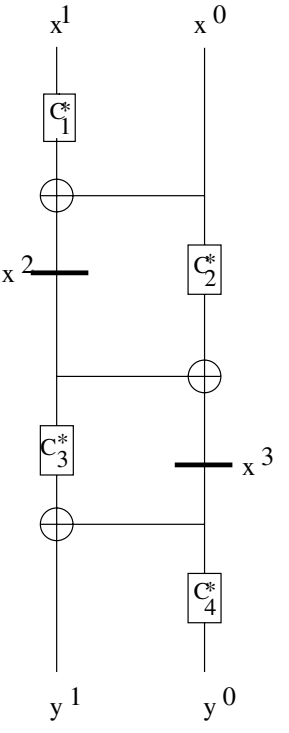


Fig. 4. L-scheme four rounds

Finally, we denote the $(x_i^0)_{i\in[1..q]}$, $(x_i^1)_{i\in[1..q]}$, $(x_i^2)_{i\in[1..q]}$, $(x_i^3)_{i\in[1..q]}$, $(y_i^0)_{i\in[1..q]}$ and $(y_i^1)_{i\in[1..q]}$ q-tuples of n-bit words by x^0, x^1, x^2, x^3, y^0, y^1 respectively.

We now define $\mathcal{X}$ as the set of X q-tuples of pairwise distinct I_{2n} words (i.e. such that for any distinct i,j numbers in $[1..q]$, $x_i^1 \neq x_j^1$ or $x_i^0 \neq x_j^0$), and define $\mathcal{Y}$ as the set of those Y q-tuples of I_{2n} words such that the corresponding y^1 q-tuples consists of pairwise distinct I_n words: $\mathcal{Y} = \{(Y_1, \cdots, Y_q) \in (I^{2n})^q \ / \ y^1 \in I^{\neq},\ y^0 \in I^{\neq}\}$.

We want to lower bound the size of $\mathcal{Y}$ and the $\Pr[X \overset{f}{\mapsto} Y]$ transition probability associated with any X q-tuple in $\mathcal{X}$ and any Y q-tuple in $\mathcal{Y}$ and show that there exists ϵ_1 and ϵ_2 real numbers satisfying conditions of Theorem 1.

We have (for more details, see section 5.1):

$$\begin{aligned} |\mathcal{Y}| &= |I^{2n}|^q \cdot (1 - \Pr[y^1 \notin I^{\neq}]) \\ &\geq |I^{2n}|^q (1 - \frac{q(q-1)}{2.2^n}) \end{aligned}$$

So, we can take $\epsilon_1 = \frac{q(q-1)}{2.2^n}$.

Now, given any X q-tuple of $\mathcal{X}$ and any Y q-tuple of $\mathcal{Y}$ let us establish a lower bound on $\Pr[X \overset{f}{\mapsto} Y]$ (for more details, see section 5.1).

$$\Pr[X \overset{f}{\mapsto} Y] \geq \sum_{x^2,x^3,x^3\oplus y^1 \in I^{\neq}} \Pr[(c_1^*(x^1) \oplus x^0 = x^2) \wedge (x^3 = c_2^*(x^0))] \oplus x^2) \cdot \Pr[(c_3^*(x^2) \oplus x^3 = y^0) \wedge (c_4^*(x^3) = y^1)] \quad (3)$$

First, for any x^2, x^3, $x^3 \oplus y^1$ q-tuple of $I^{\neq}$, we have $\Pr[(c_3^*(x^2) \oplus x^3 = y^0) \wedge (c_3^*(x^3) = y^1)] = \left(\frac{(2^n-q)!}{2^n!}\right)^2 \geq \frac{1}{2^{2nq}}$ (for more details, see section 5.1). Therefore, inequality (1) implies:

$$\Pr[X \overset{f}{\mapsto} Y] \geq \sum_{x^2,x^3,x^3\oplus y^1 \in I^{\neq}} \frac{1}{2^{2nq}} \cdot \Pr[(c_1^*(x^1) \oplus x^0 = x^2) \wedge (c_1^*(x^1) \oplus y^0 = x^3)] \quad (i)$$

Let us now estimate:

$$B = \sum_{x^2,x^3,x^3\oplus y^1 \in I^{\neq}} \Pr[(c_1^*(x^1) \oplus x^0 = x^2) \wedge (c_2^*(x^0) \oplus x^2 = x^3)]$$

We have:

$$\begin{aligned} B &= \Pr[(c_1^*(x^1) \oplus x^0) \in I^{\neq} \wedge (c_2^*(x^0) \oplus c_1^*(x^1) \oplus x^0) \in I^{\neq} \wedge (x^3 \oplus y^1) \in I^{\neq}] \\ &= 1 - \Pr[(c_1^*(x^1) \oplus x^0 \in I^{=}) \vee (c_2^*(x^0) \oplus c_1^*(x^1) \oplus x^0) \in I^{=} \vee (x^3 \oplus y^1) \in I^{=}] \\ &\geq 1 - 3.\frac{q(q-1)}{2}.\frac{2}{2^n} \end{aligned}$$

By using inequalities (i) and (ii), we obtain:

$$\Pr[X \overset{f}{\mapsto} Y] \geq \left(1 - \frac{3q(q-1)}{2^n}\right) \cdot \frac{1}{2^{2nq}}$$

We can notice that $Pr[X \overset{f^*}{\mapsto} Y] = \frac{1}{2^{2nq}}$. So we can apply Theorem 1 with $\epsilon_1 = \frac{q(q-1)}{2|I^n|}$ and $\epsilon_2 = \frac{3q(q-1)}{|I^n|}$. We obtain:

$$\begin{aligned} \mathrm{Adv}_{\mathcal{A}}^q(f, f^*) &\leq \tfrac{7q(q-1)}{2.2^n} \\ \mathrm{Adv}_{\mathcal{A}}^q(f, f^*) &\leq \tfrac{7q^2}{2.2^n} \end{aligned}$$

□

References

[Ai96] W. Aiello, R. Venkatesan, "Foiling Birthday Attacks in Length-Doubling Transformations". In *Advances in Cryptology - Eurocrypt'96*, LNCS 1070, p. 307, Springer Verlag, Saragossa, Spain, May 1996.

[Be94] M. Bellare, J. Kilian, P. Rogaway, "The Security of Cipher Block Chaining". In *Advances in Cryptology - CRYPTO'94*,LNCS 839, p. 341, Springer-Verlag, Santa Barbara, U.S.A., 1994.

[Ka] Specification of the 3GPP confidentiality and Integrity algorithm KASUMI. Documentation available on http://www.etsi.org/

[Ka97] Y. Kaneko, F. Sano, K. Sakurai, "On Provable Security against Differential and Linear Cryptanalysis in Generalized Feistel Ciphers with Multiple Random Functions". In *Selected Areas in Cryptography - SAC'97*, Ottawa, Canada, August 1997.

[La90] X. Lai, J.L. Massey, "A Proposal for a New Block Encryption Standard". In *Advances in Cryptology - Eurocrypt'90*, LNCS 473 , p. 389, Springer Verlag, Aarhus, Denemark, 1991.

[Lu88] M. Luby, C. Rackoff, "How to Construct Pseudorandom Permutations from Pseudorandom Function". In *Siam Journal on Computing* , vol. 17, p. 373, 1988.

[Ma92] U. Maurer, "A Simplified and generalised treatment of Luby-Rackoff Pseudorandom Permutation Generators", In *Advances in Cryptology - Eurocrypt'92*, LNCS 658 , p. 239, Springer Verlag, New York, USA, 1992.

[Ma93] M. Matsui, "New Block Encryption Algorithm MISTY", In *Fast Software Encryption - FSE'97*, LNCS 1267, p. 54, Springer Verlag, Haifa, Israel, 1997.

[Pa91] J. Patarin, "Etude de Générateurs de Permutation Basés sur le Schéma du D.E.S. ", Phd. Thesis, University of Paris VI, 1991.

[Sa97] K. Sakurai, Y. Zheng, "On Non-Pseudorandomness from Block Ciphers with Provable Immunity Against Linear Cryptanalysis, In *IEICE Trans. Fundamentals*, vol. E80-A, n. 1, January 1997.

[Su96] M. Sugita, "Pseudorandomness of a Block Cipher MISTY", Technical Reporrt of IEICE, ISEC96-9.

[Su97] M. Sugita, "Pseudorandomness of a Block Cipher with Recursive Structures", Technical Report of IEICE, ISEC97-9.

[Va99] S. Vaudenay, "On Provable Security for Conventional Cryptography", In *ICISC'99*, invited lecture.

[Zh89] Y. Zheng, T. Matsumoto, H. Imai, "On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses". In *Advances in Cryptology - CRYPTO'89*, LNCS 435, p. 461, Springer-Verlag, Santa Barbara, U.S.A., 1990.

NESSIE: A European Approach to Evaluate Cryptographic Algorithms

Bart Preneel

Katholieke Univ. Leuven, Dept. Electrical Engineering-ESAT,
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
bart.preneel@esat.kuleuven.ac.be

Abstract. The NESSIE project (New European Schemes for Signature, Integrity and Encryption) intends to put forward a portfolio containing the next generation of cryptographic primitives. These primitives will offer a higher security level than existing primitives, and/or will offer a higher confidence level, built up by an open evaluation process. Moreover, they should be better suited for the constraints of future hardware and software environments. In order to reach this goal, the project has launched an open call in March 2000. In response to this call, 39 primitives have been submitted by September 29, 2000, many of these from major players. Currently, the NESSIE evaluation process is under way; it considers both security and performance aspects. This article presents the status of the NESSIE project after 15 months.

1 Introduction

NESSIE is a research project within the Information Societies Technology (IST) Programme of the European Commission. The participants of the project are:

- Katholieke Universiteit Leuven (Belgium) (coordinator);
- Ecole Normale Supérieure (France);
- Royal Holloway, University of London (U.K.);
- Siemens Aktiengesellschaft (Germany);
- Technion – Israel Institute of Technology (Israel);
- Université Catholique de Louvain (Belgium); and
- Universitetet i Bergen (Norway).

NESSIE is a 3-year project, which started on 1st January 2000. This paper intends to present the state of the project in April 2000. It is organised as follows. Section 2 presents the NESSIE call and its results. Section 3 discusses the tools which the project is developing to support the evaluation process. Sections 4 and 5 deal with the security and performance evaluation respectively. Section 6 raises some intellectual property issues. The NESSIE approach towards dissemination and standardisation is presented in Sect. 7. Finally, conclusions are put forward in Sect. 8.

Detailed and up to date information on the NESSIE project is available at the project web site: http://cryptonessie.org/.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 267–276, 2002.

2 NESSIE Call

In the first year of the project, an open call for the submission of cryptographic primitives, as well as for evaluation methodologies for these primitives has been launched. The scope of this call has been defined together with the project industry board (PIB) (cf. Sect. 7), and it was published in March 2000. The deadline for submissions was 29 September 2000. In response to this call NESSIE received 40 submissions, all of which met the submission requirements.

2.1 Contents of the NESSIE Call

The NESSIE call includes a request for a broad set of primitives providing confidentiality, data integrity, and authentication. These primitives include block ciphers, stream ciphers, hash functions, MAC algorithms, digital signature schemes, and public-key encryption and identification schemes (for definitions of these primitives, see [17]). In this respect the NESSIE call is very similar to the two calls of RIPE (Race Integrity Primitives Evaluation, 1988-1992) [20]. In addition, the NESSIE call asks for evaluation methodologies for these primitives.

It has been stressed that not only block ciphers are needed, but also several other primitives such as stream ciphers, MACs, etc. (since there are many block ciphers around but far fewer other primitives even though they are required in many applications). Thus the scope of this call is much wider than that of the AES call launched by NIST in 1997.

The call also specifies the main selection criteria which will be used to evaluate the proposals. These criteria are long-term security, market requirements, efficiency and flexibility. Primitives can be targeted towards a specific environment (such as 8-bit smart cards or high-end 64-bit processors), but it is clearly an advantage to offer a wide flexibility of use. Security is put forward as the most important criterion, as security of a cryptographic primitive is essential to achieve confidence and to build consensus.

For the *security requirements* of symmetric primitives, two main security levels are specified, named *normal* and *high*. The minimal requirements for a symmetric primitive to attain either the normal or high security level depend on the key length, internal memory, or output length of the primitive. For block ciphers a third security level, *normal-legacy*, is specified, with a block size of 64 bits compared to 128 bits for the normal and high security level. The motivation for this request are applications such as UMTS/3GPP, which intend to use 64-bit block ciphers for the next 10-15 years. Remark that the AES call only specifies a 128-bit block size. For the asymmetric primitives the security level is specified in terms of the computational effort of the most efficient attack.

If selected by NESSIE, the primitive should preferably be available royalty-free. If this is not possible, then access should be non-discriminatory. The submitter should state the position concerning intellectual property and should update it when necessary.

The submission requirements are much less stringent than for AES, particularly in terms of the requirement for software implementations (only 'portable C' is mandatory).

2.2 Response to the NESSIE Call

The cryptographic community has responded very enthusiastically to the call. Thirty nine primitives have been received, as well as one proposal for a testing methodology. After an interaction process, which took about one month, all submissions comply with the requirements of the call. The results of the call seem promising, and will lead to a very interesting and challenging evaluation process. There are 26 symmetric primitives:

- seventeen block ciphers, which is probably not a surprise given the increased attention to block cipher design and evaluation as a consequence of the AES competition organised by NIST. They are divided as follows:
  - six 64-bit block ciphers: CS-Cipher, Hierocrypt-L1, IDEA, Khazad, MISTY1, and Nimbus;
  - seven 128-bit block ciphers: Anubis, Camellia, Grand Cru, Hierocrypt-3, Noekeon, Q, and SC2000 (none of these seven come from the AES process);
  - one 160-bit block cipher: Shacal; and
  - three block ciphers with a variable block length: NUSH (64, 128, and 256 bits), RC6 (at least 128 bits), and SAFER++ (64 and 128 bits).
- six synchronous stream ciphers: BMGL, Leviathan, LILI-128, SNOW, SOBER-t16, and SOBER-t32.
- two MAC algorithms: Two-Track-MAC and UMAC; and
- one collision-resistant hash function: Whirlpool.

Thirteen asymmetric primitives have been submitted:

- five asymmetric encryption schemes: ACE Encrypt, ECIES, EPOC, PSEC, and RSA-OAEP;
- seven digital signature algorithms: ACE Sign, ECDSA, ESIGN, FLASH, QUARTZ, RSA-PSS, and SFLASH; and
- one identification scheme: GPS.

Approximately[1] seventeen submissions originated within Europe (6 from France, 4 from Belgium, 3 from Switzerland, 2 from Sweden), nine in North America (7 USA, 2 from Canada), nine in Asia (8 from Japan), three in Australia and three in South America (Brazil). The majority of submissions originated within industry (27); seven came from academia, and six are the result of a joint effort between industry and academia. Note however that the submitter of the algorithm may not be the inventor, hence the share of academic research is probably underestimated by these numbers.

On 13–14 November 2000 the first NESSIE workshop was organised in Leuven (Belgium), where 35 submissions were presented. All submissions are available on the NESSIE web site.

[1] Fractional numbers have been used to take into account primitives with submitters over several continents/countries – the totals here are approximations by integers, hence they do not add up to 40.

3 Tools

It would be rather naive to believe that modern computers and sophisticated software tools can replace human cryptanalysis. Nevertheless, software plays an important role in modern cryptanalysis. In most cases, the attacks found by the cryptanalyst require a large number of computational steps, hence the actual execution of the attack is done on a computer. However, software and software tools can also be essential to find a successful way to attack a cryptographic algorithm; examples include differential and linear cryptanalysis, dependence tests, and statistical tests.

Within NESSIE, we distinguish two classes of tools. The general tools are not specific for the algorithms to be analysed. Special tools, which are specific for the analysis of one algorithm, will be implemented when, in the course of the cryptanalysis of an algorithm, the need for such a tool turns up.

For the evaluation of the NESSIE submissions, a comprehensive set of general tools is available within the project. These tools are in part based on an improved version of the tools developed by the RIPE (RACE Integrity Primitives Evaluation) project [20]. These test include: the frequency test, the collision test, the overlapping m-tuple test, the gap test, the constant runs test, the coupon collector's test, Maurer's universal test [16], the poker test, the spectral test, the correlation test, the rank test, the linear, non-linear, and dyadic complexity test, the Ziv-Lempel complexity test, the dependence test, the percolation test, the linear equation, linear approximation and correlation immunity test, the linear factors test, and a cycle detection tool.

The NESSIE project is also developing a new generic tool to analyse block ciphers with differential [5] and linear cryptanalysis [15]. This tool is based on a general description language for block ciphers. Other tools under development comprise tools for related key attacks [3] and tools for χ^2 cryptanalysis [21].

In September 2000, the US NIST published a suite of statistical tests for the evaluation of sequences of random or pseudorandom bits; this document has been revised in December 2000 [19]. A careful comparison will be made between the RIPE and NIST test suites.

The software for these tools will not be made available outside the project, but all the results obtained using these tools will be made public in full detail.

4 Security Evaluation

4.1 Methodology

We first describe the internal process within NESSIE used to assess submissions. Initially each submission was assigned to a NESSIE partner. The NESSIE partner performed basic checks on the submission, such as compliance with the call, working software, obvious weaknesses etc. The aim of this initial check was mainly to ensure that submissions were specified in a consistent and cogent form in time for the November 2000 workshop. It is vital for proper security assessments that the algorithms are fully and unambiguously described. This process

did require interaction with some submitters to ensure that the submissions were in the required form.

The next internal stage (November 2000) was to assign each submission to a pair of NESSIE partners for an initial detailed evaluation. Each submission has then been subject to two independent initial assessments. After the two initial assessments of a submission have taken place, the two NESSIE partners have produced a joint summary of their assessments concerning that submission. Clearly, any future assessment of a submission will depend on this joint summary, but it is generally anticipated that the two NESSIE partners would then further jointly assess the submission.

It is anticipated that the report published at the end of the first phase of the security evaluation (August 2001) will contain a security assessment of the primitives against standard cryptanalytic techniques, the results of statistical testing (where appropriate) and an indication of types of future assessment to be carried out. This preliminary assessment will include a rationale for submissions rejected at this stage.

4.2 First Results

In the short time available for analysis so far, there have been significant results concerning some submissions. The analyses of the block ciphers Q [4] and Nimbus [12] and of the next bit testing methodology already indicate that they are not suitable for recommendation by the NESSIE project. The 'SQUARE' type attack on Hierocrypt-3 and Hierocrypt-L1 described by the designers has been improved [2]. There are also some external preliminary results concerning Leviathan [8], LILI-128 [1], and NUSH. However, it should be stressed that at the time of writing of this paper (March 2001), the assessment process is at a very early stage.

5 Performance Evaluation

5.1 Background and Motivation

Performance evaluation is an essential part in the assessment of a cryptographic algorithm: efficiency is a very important criterion in deciding for the adoption of an algorithm.

The candidates will be used on several platforms (PCs, smart cards, dedicated hardware) and for various applications. Some applications have tight timing constraints (e.g., banking applications, cellular phones); for other applications a high throughput is essential (e.g., high speed networking, hard disk encryption).

5.2 The Methodology

First a framework has been defined to compare the performance of primitives on a fair and equal basis. It will be used for all evaluations of submitted candidates or asymmetric primitives, for block ciphers as well as stream ciphers, hash functions, etc.

First of all a theoretical approach has been established. For each candidate we will dissect the algorithm into three parts: setup (independent of key and data), precomputations (independent of data, e.g., key schedule) and the primitive itself (that must be repeated for every use).

Next a set of four test platforms has been defined on which each candidate may be tested. These platforms are smart cards, 32-bit PCs, 64-bit machines, and Field Programmable Gate Arrays (FPGAs).

Then rules have been defined which specify how performance should be measured on PCs, smart cards, 64-bit machines and FPGAs. The implementations parameters depend on the platform, but may include RAM, speed, code size, chip area, and power consumption. On smart cards, only the following parameters will be taken into account, in decreasing order of importance: RAM usage, speed, code size. On PCs, RAM has very little impact, and speed is the main concern. On FPGAs, throughput, latency, chip area and power consumption will be considered. Unfortunately, the limited resources of the project will not allow for the evaluation of dedicated hardware implementations (ASICs), but it may well be that teams outside the project can offer assistance for certain primitives.

The project will also consider the resistance of implementations to physical attacks such as timing attacks [13], fault analysis [6,7], and power analysis [14].

For non constant-time algorithms (data or key dependence, asymmetry between encryption and decryption) the data or key dependence will be analysed; other elements that will be taken into account include the difference between encryption and decryption, and between signature and verification operation. For symmetric primitives, the key agility will also be considered.

This approach will result in the definition of platform dependent test and several platform dependent rekeying scenarios. Low-cost smart cards will only be used for block ciphers, MACs, hash functions, stream ciphers, pseudorandom number generation, and identification schemes.

5.3 Template for Performance Evaluation

In order to present performance information in a consistent way within the NESSIE project, a performance 'template' has been developed. The goal of this template is to collect intrinsic information related to the performance of the submitted candidates. It is presented as a questionnaire, which can be completed by analyzing the primitive and the software for the primitive. A first part describes parameters such as word size, memory requirement, key size and code size. Next the basic operations are analysed, such as shift/rotations, table look-ups, permutations, multiplications, additions, modular reduction, exponentiation, inversion,.... Then the nature and speed of precomputations (setup, key schedule, etc.) is described. Elements such as the dependence on the keys and on the inputs determine whether the code is constant-time or not.

6 Intellectual Property

An important element of the call is the intellectual property statement of the submitters. While it were ideal for users of the NESSIE results that all algorithms

recommended by NESSIE were in the public domain, it is clear that this is for the time being not realistic. The users in the NESSIE PIB have clearly stated that they prefer to see royalty-free primitives, preferably combined with open source implementations. However, providers of intellectual property, who are also represented in the PIB, typically have different opinions.

One observation is that in the past, there has always been a very large difference between symmetric and asymmetric cryptographic primitives. Therefore it is not so surprising that NIST was able to require that the designers of the block cipher selected for the AES would give away all their rights, if their algorithm was selected; it is clear that this is not a realistic expectation for the NESSIE project.

In this section we will attempt to summarise the intellectual property statements of the submitters. Note however that this interpretation is only indicative; for the final answer the reader is referred to the intellectual property statement on the NESSIE web page, and to the submitters themselves.

Nineteen out of 39 submissions are in the public domain, or the submitters indicate that a royalty-free license will be given. These are the block ciphers Anubis, CS-Cipher, Grand Cru, Khazad, Misty1, Nimbus, Noekeon, Nush, Q, Shacal, Safer++, the stream ciphers BMGL, LILI-128 and SNOW, the MAC algorithms Two-Track-MAC and UMAC, the hash function Whirlpool, and the public-key primitives RSA-OAEP[2] (public-key encryption) and RSA-PSS[2] (digital signature scheme).

The block cipher IDEA is free for non-commercial use only; for commercial applications a license is required. The stream ciphers SOBER-t16 and SOBER-t32 are royalty-free in non-embedded applications, but licenses are required for embedded applications.

Licenses under reasonable and non-discriminatory terms will be given for the block ciphers Camellia, Hierocrypt-L1, Hierocrypt-L3 and SC2000, the stream cipher Leviathan[3], and for the public-key encryption algorithms EPOC and PSEC and the digital signature scheme ESIGN. Similar conditions hold for ACE Crypt and ACE Sign, but in this case the detailed license conditions are rather complex (that is, they cannot be summarised in a few words). Additions to the 'reasonable and non-discriminatory' terms are required for the public-key primitives ECDSA and ECIES; it is required that the license holder reciprocates some of his rights.

For the digital signature schemes FLASH, SFLASH and QUARTZ the licensing conditions are expected to be non-discriminatory, but no decision has been made yet. A similar statement holds for the identification scheme GPS, but in this case certain applications in France may be excluded from the license.

Finally, the submitters of RC6 are willing to negotiate licenses on reasonable terms and conditions.

It is clear that intellectual property is always a complex issue, and it will not be possible to resolve this completely within the framework of NESSIE. The call

[2] This statement does not hold for the variants of RSA with more than two primes.
[3] If this algorithm is recommended by NESSIE.

for papers for the second NESSIE workshop will definitely invite comments on this issue.

7 Dissemination and Standardisation

7.1 An Open Evaluation Process

The NESSIE project intends to be an open project, which implies that the members of the public are invited to contribute to the evaluation process. In order to facilitate this process, all submissions are available on the NESSIE website, and comments are distributed through this website. The NESSIE website contains an open forum where everyone can post contributions on the primitives and on the process. In addition, three open workshops are organised during the project; the first one has taken place in November 2000. The second workshop has been scheduled for September 12-13, 2001 (University of London, Egham, UK), and the third one for the Fall of 2002.

7.2 The Project Industry Board

The Project Industry Board (PIB) was established to ensure that the project addresses real needs and requirements of industry dealing with the provision and use of cryptographic techniques and cryptographic products. Two meetings are planned per year, but additional meetings may be held to address specific issues or concerns that may arise.

Membership was originally by invitation, but there have been a number of subsequent requests to join the PIB. Currently it consists of about twenty-five leading companies which are users or suppliers of cryptology. During the first year, the PIB has provided very useful input to all aspects of the NESSIE project.

7.3 Standardisation

Together with the NESSIE PIB, the project will establish a standardisation strategy. It is not our intention to establish a new standardisation body or mechanism, but to channel the NESSIE results to the appropriate standardisation bodies, such as, ISO/IEC, IETF, IEEE and EESSI (European Electronic Signature Standardisation Initiative). We believe that the NESSIE approach of open evaluation is complementary to the approach taken by standardisation bodies. Indeed, these bodies typically do not have the resources to perform any substantial security evaluation, which may be one of the reasons why standardisation in security progresses often more slowly than anticipated.

The NESSIE project will also take into account existing and emerging standards, even if these have not been formally submitted to the NESSIE project. Two recent examples in this context come from the standardisation efforts run by NIST. The NESSIE project has contributed to the AES process, and one of the designers of the AES algorithm 'Rijndael' [9,11] is a member of the NESSIE project team. It is therefore clear that in the evaluation of block ciphers, the

Rijndael algorithm will be used as a benchmark. The NESSIE project will also study the security and performance of SHA-2 [18], the new hash algorithm proposed by NIST to extend the result of SHA-1 [10] to hash results between 256 and 512 bits.

8 Conclusion

We believe that the approach of the NESSIE project provides an interesting approach to provide the users with the next generation of cryptographic algorithms, and to stimulate further research on cryptographic algorithms.

The NESSIE evaluation process will be delicate and complex, but we are confident that the project will be able to live up to the expectations. In the first months of the evaluation, several research papers have been written by project members describing cryptanalytic results of the NESSIE project, and several comments were received from other cryptographers.

We invite the readers of this article to contribute towards the NESSIE project by sending comments on individual algorithms, on the evaluation process, and on their expectations towards the project results.

Acknowledgments. I would like to thank all the members of the NESSIE project, and more in particular the contributors to the first annual report, on which this text is largely based: Eli Biham, Antoon Bosselaers, Mathieu Ciet, Markus Dichtl, Louis Granboulan, Keith Howker, Lars Knudsen, Sean Murphy, François Koeune, and Francesco Sica.

The work described in this paper has been supported by the Commission of the European Communities through the IST Programme under Contract IST–1999–12324.

Disclaimer

The information in this paper is provided as is, and no guarantee or warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

References

1. S. Babbage, "Cryptanalysis of LILI-128," Preprint, 2001.
2. P. Barreto, V. Rijmen, J. Nakahara Jr., B. Preneel, J. Vandewalle, H. Kim, "Improved Square attacks against reduced-round Hierocrypt," *Preproceedings Fast Software Encryption 2001*, M. Matsui, Ed.
3. E. Biham, "New types of cryptanalytic attacks using related keys," *Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 398–409.
4. E. Biham, V. Furman, M. Misztal, V. Rijmen, "Differential Cryptanalysis of Q," *Preproceedings Fast Software Encryption 2001*, M. Matsui, Ed.

5. E. Biham, A. Shamir, *"Differential Cryptanalysis of the Data Encryption Standard,"* Springer-Verlag, 1993.
6. E. Biham, A. Shamir, "Differential fault analysis of secret key cryptosystems," *Advances in Cryptology, Proceedings Crypto'97, LNCS 1294*, B. Kaliski, Ed., Springer-Verlag, 1997, pp. 513–525.
7. D. Boneh, R. A. DeMillo, R. J. Lipton, "On the importance of checking cryptographic protocols for faults," *Advances in Cryptology, Proceedings Eurocrypt'97, LNCS 1233*, W. Fumy, Ed., Springer-Verlag, 1997, pp. 37–51.
8. P. Crowley, S. Lucks, "Bias in the Leviathan stream cipher," *Preproceedings Fast Software Encryption 2001*, M. Matsui, Ed.
9. J. Daemen, V. Rijmen, "AES proposal Rijndael," September 3, 1999, available from `http://www.nist.gov/aes`.
10. FIPS 180-1, *"Secure Hash Standard,"* Federal Information Processing Standard (FIPS), Publication 180-1, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., April 17, 1995.
11. FIPS XXX *"Advanced Encryption Standard (AES),"* Washington D.C.: NIST, US Department of Commerce, Draft, February 28, 2001.
12. V. Furman, "Differential cryptanalysis of Nimbus," *Preproceedings Fast Software Encryption 2001*, M. Matsui, Ed.
13. P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 104–113.
14. P. Kocher, J. Jaffe, B. Jun, "Differential power analysis," *Advances in Cryptology, Proceedings Crypto'99, LNCS 1666*, M.J. Wiener, Ed., Springer-Verlag, 1999, pp. 388–397.
15. M. Matsui, "The first experimental cryptanalysis of the Data Encryption Standard," *Advances in Cryptology, Proceedings Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 1–11.
16. U.M. Maurer, "A universal statistical test for random bit generators," *Advances in Cryptology, Proceedings Crypto'90, LNCS 537*, S. Vanstone, Ed., Springer-Verlag, 1991, pp. 409–420.
17. A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *"Handbook of Applied Cryptography,"* CRC Press, 1997.
18. NIST, *"SHA-256, SHA-384, SHA-512,"* Washington D.C.: NIST, US Department of Commerce, Draft, 2000.
19. NIST, *"A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications,"* NIST Special Publication 800-22, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., December 2000.
20. RIPE, *"Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040),"* *LNCS 1007*, A. Bosselaers, B. Preneel, Eds., Springer-Verlag, 1995.
21. S. Vaudenay, "An experiment on DES – statistical cryptanalysis," *Proceedings 1996 ACM Conference on Computer and Communications Security*, March 14-15, New Delhi, India, 1996, pp. 139–147.

Related Key Attacks on Reduced Round KASUMI

Mark Blunden[1]* and Adrian Escott[2]*

[1] Serco Technology, UK.
mark.blunden@talk21.com
[2] Hutchison 3G, UK.
Adrian.Escott@Hutchison3G.com

Abstract. This paper describes related key attacks on five and six round KASUMI. The five round attack requires the encryption of approximately 2^{19} chosen plaintext pairs X and X^* under keys K and K^* respectively where K and K^* differ in only one bit, and requires a maximum of a little over 2^{33} trials to recover the entire key. The six round attack requires a smaller number of chosen plaintext encryptions than the five round attack, and recovers the entire key in a maximum of 2^{112} trials.

1 Introduction

KASUMI [4] is an eight round, 64-bit block cipher with a 128-bit key. It is based upon MISTY1 [7], and was designed to form the basis of the 3GPP (3rd Generation Partnership Project) confidentiality and integrity algorithms. 3GPP is the body standardizing the next generation of mobile telephony.

This paper describes differential related key attacks on five and six rounds of KASUMI. Differential attacks were introduced by Biham and Shamir in [3]. Related key attacks were first introduced by Biham [2], although a similar idea can be found in Knudsen [6]. Differential related key attacks are discussed in Kelsey *et al* [5].

The attacks require the encryption of plaintext pairs X and X^* under keys K and K^* respectively, where K and K^* differ in only one bit. The attack on five round KASUMI requires on average 2^{19} chosen plaintext pairs, and a maximum of a little over 2^{33} trials to find the entire key. The six round attack requires the encryption of on average $3 \cdot 2^{17}$ chosen plaintext pairs, and a maximum of 2^{112} trials to find the entire key.

Note that the attacks described in this paper present no security threat to KASUMI as used in 3GPP as firstly the attacks do not cover all rounds of KASUMI, and secondly it should not be possible to manipulate the keys in the required way within the 3GPP environment. The motivation for investigating such attacks comes from the fact that one significant difference between KASUMI and MISTY1 is the design of the key schedule, with the key schedule of KASUMI being much simpler than that of MISTY1.

* Part of this work was done while the authors were with BTexaCT, UK.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 277–285, 2002.

2 KASUMI

KASUMI shares with MISTY1 the design goals of having a numerical basis for its security and of being sufficiently fast when implemented in hardware. More details of the design rationale of KASUMI can be found in [1]. To meet the first goal, the design of KASUMI is based on the theory of provable security of block ciphers against linear and differential cryptanalysis introduced by Nyberg and Knudsen [8][9]. To meet the second design goal, both the S-boxes and the key schedule were carefully chosen to optimize the hardware performance.

KASUMI is an eight round Fiestel type cipher. Each round is made up of an FL function and an FO function. In odd numbered rounds the FL function precedes the FO function, whereas in even numbered rounds the FO function precedes the FL function. The FL function is a simple function that is linear for a fixed key. The FO function contains the non-linearity in each round. The FO function is made up from three FI functions, which in turn are made up from two applications of two different S-boxes and a key addition. The S-boxes are chosen to have optimal resistance to linear and differential cryptanalysis. The overall structure is shown in Figure 1.

KASUMI has a simple, linear key schedule in order to make the hardware significantly smaller and to reduce key set-up time. Every bit of the key is used once in every round. For the purposes of this paper it is sufficient to consider every ith round key as being made up of three parts KL_i, KO_i and KI_i. These in turn are divided into a total of eight 16-bit parts such that $KL_i = KL_{i,1} \,\|\, KL_{i,2}$, $KO_i = KO_{i,1} \,\|\, KO_{i,2} \,\|\, KO_{i,3}$ and $KI_i = KI_{i,1} \,\|\, KI_{i,2} \,\|\, KI_{i,3}$, where $\|$ denotes concatenation. The $KI_{i,j}$ are further divided into two parts (first one of seven bits, second of nine) with $KI_{i,j} = KI_{i,j,1} \,\|\, KI_{i,j,2}$. The round keys are derived by splitting the key K into eight 16-bits parts $K_1 \,\|\, \cdots \,\|\, K_8$. Each part of the key is used to derive exactly one round key part in each round. The key schedule has the property that changing one bit of K changes exactly one key bit of each round key.

3 Four Round Differentials of KASUMI

This section describes a family of four round differentials that can be used to attack up to six rounds of KASUMI. The core idea behind this family of differentials is to choose small key differences and control the effect of these differences as they progress through the cipher using either the properties of the FL function or appropriately chosen plaintext differences.

The differential is formed by encrypting plaintexts X and X^* under keys K and K^* respectively, where $K_j = K_j^*$ for $j = 1, 2, 4, \ldots, 8$, $K_3 = K_3^* \oplus k$ for any 16-bit string k, $X = 0^{16}1^{16}a$ for any 32-bit string a, $X^* = 0^{16}1^{16}(a \oplus (k <<< 5)0^{16})$ and $<<< n$ denotes a left circular rotation by n bits. With probability of $2^{-wt(k)}$, where $wt(k)$ is the number of non-zero bits of k, the output difference at the end of the fourth round is $c(k <<< 5)0^{16}$, where c is any 32 bit string. In practice, k is chosen such that $wt(k) = 1$ in order to maximize the probability

of the differential holding. The following paragraph provides more detail as to why the differential works.

If K and K^* are as defined above, then the only non zero round key part differences in the first three rounds are $\Delta KL_{1,2} = k$, $\Delta KO_{2,1} = k <<< 5$ and $\Delta KL_{3,1} = k <<< 1$. Now, the FL function has the property that for any key KL, an input of $0^{16}1^{16}$ always gives an output of 1^{32}. Hence when encrypting plaintexts X and X^* as defined above under keys K and K^* respectively, the input and output differences of $FL1$ are both 0^{32}. The round key difference in

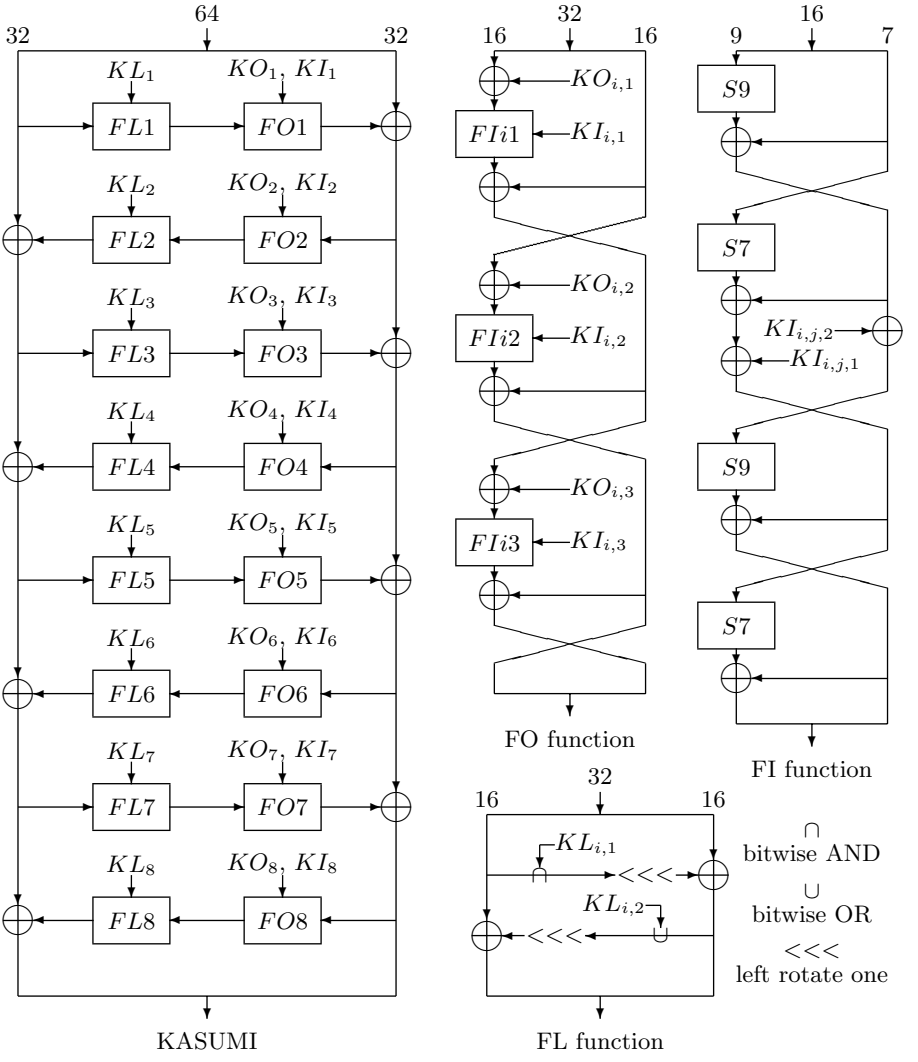


Fig. 1. KASUMI

the second round combines with the difference in the right hand side of the plaintexts such that the output difference of $FO2$ is always 0^{32}. Since the input difference to $FL3$ is 0^{32}, and the AND of a message bit and key bit is always zero when the message bit is zero, the difference in the third round key has no effect on the output of $FL3$ with probability $2^{-wt(k)}$. Therefore, with probability $2^{-wt(k)}$ the output difference at the end of the fourth round is $c(k <<< 5)0^{16}$.

Figure 2 illustrates this family of four round differentials over six round KASUMI.

Note that the restriction on the inputs to $FL1$ to always be of the form $0^{16}1^{16}$ is stronger than necessary when $wt(k) < 16$, but is defined this way for simplicity.

4 An Attack on Five Round KASUMI

This section describes a related key attack on five round KASUMI. The attack requires an average of 2^{15} chosen plaintext encryptions under the original key and 2^{19} chosen plaintext encryptions under a chosen related key. The attack is expected to recover the entire key in a maximum of a little over 2^{33} trials, where each trial is equivalent to approximately $1{\cdot}6$ five round KASUMI encryptions.

When encrypting a plaintext pair X and X^* under keys K and K^* respectively, where X, X^*, K and K^* are as described in Section 3, with probability 2^{-16} the ciphertext difference is $Y \oplus Y^* = cge$, where c is any 32 bit string, e is any 16 bit string and $g = e \oplus (k <<< 5)$. When $wt(k) = 1$ the attack requires an average of eight ciphertext pairs of this form, and six ciphertext pairs whose inputs are of this form but whose output difference can be of any form. It is expected that these ciphertext pairs can be found from a set of 2^{19} ciphertext pairs generated from inputs of the appropriate form. One method of generating this set is to obtain the encryptions of 2^{15} chosen plaintexts under the key K and for each such encryption obtain the sixteen related encryptions that result from using the different values of k such that $wt(k) = 1$. Note that with K and K^* as defined above, the only non zero round key part difference in round five is $\Delta KO_{5,3} = k <<< 13$. Note also that the rightmost 32 bits of the plaintexts do not need to be known.

The attack assumes that the eight ciphertext pairs whose output differences are of the correct form all satisfy the differential described in Section 3. Consequently, for each of these eight ciphertext pairs it follows that the output difference of $FO5$ is $ge \oplus (k <<< 5)0^{16} = ee$, so that the output difference of $FI53$ is $e \oplus e = 0^{16}$. Since $\Delta KI_{5,3} = 0^{16}$, the input difference to $FI53$ is also zero. Therefore, as $\Delta KO_{5,3} = k <<< 13$, the output difference of $FI52$ is $e \oplus (k <<< 13)$. Furthermore, for each guess of $KL_{5,1}$ and $KO_{5,2}$, the inputs to $FI52$ are known. Hence, for each ciphertext pair, the difference tables of $S7$ and $S9$ can be used to suggest values of $KI_{5,2}$.

For each guess of $KL_{5,1}$ and $KO_{5,2}$, a count is made of the number of times each possible value of $KI_{5,2}$ is suggested by a ciphertext pair. The number of times a key $KI_{5,2}$ is suggested for a particular choice of $KL_{5,1}$ and $KO_{5,2}$ is

taken as the number of times the key triple $KL_{5,1}$, $KO_{5,2}$, $KI_{5,2}$ is suggested. Any key triple suggested as many as four times is kept.

The correct value of the triple $KL_{5,1}$, $KO_{5,2}$ and $KI_{5,2}$ will be included in the values suggested by each of the ciphertext pairs that do genuinely follow the differential of Section 3. Such pairs are referred to as right pairs. Since it

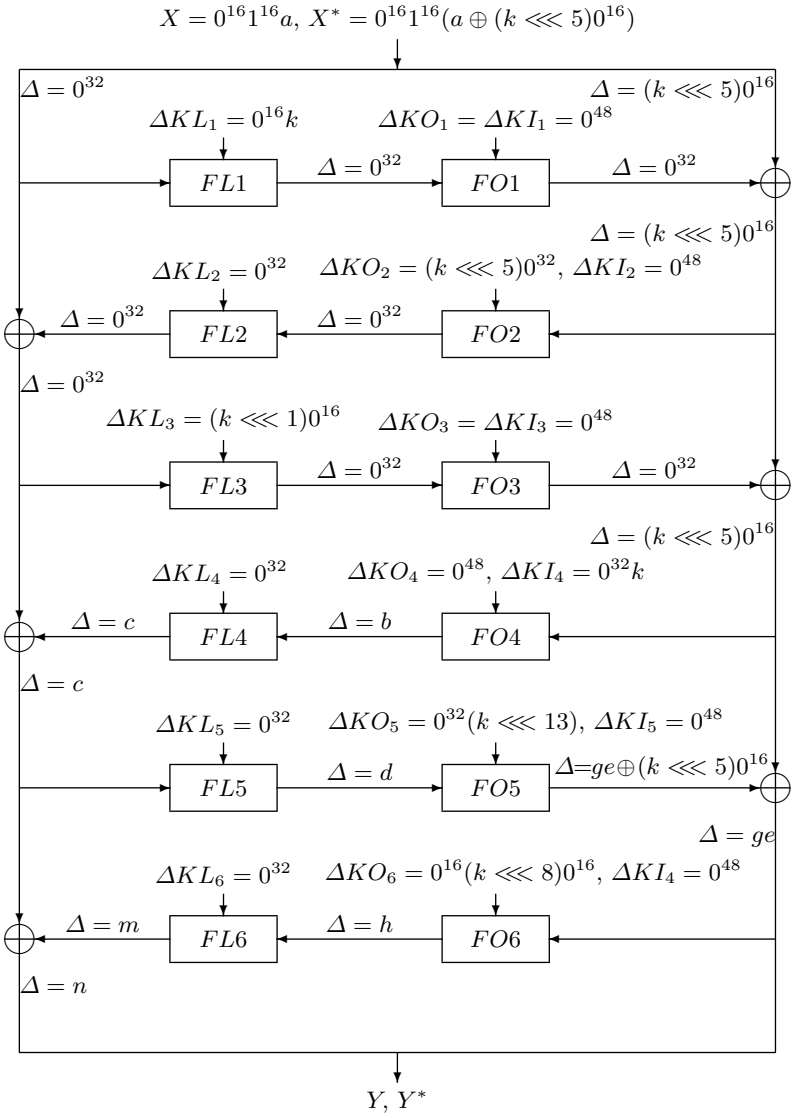


Fig. 2. A family of four round differentials over six-round KASUMI which hold with probability one half, where k, e and g are any 16-bit strings and a, b, c, d, h and m are any 32 bits strings.

is expected that four of the eight pairs satisfy the differential, the correct value of the triple $KL_{5,1}$, $KO_{5,2}$ and $KI_{5,2}$ is expected to be suggested at least four times. The probability of an incorrect triple being suggested by one ciphertext pair is 2^{-16}, and the probability of an incorrect triple being suggested as many as four times is less than 2^{-57}. Therefore, the probability that at least one incorrect triple is suggested as many as four times is less than 2^{-9}. Thus the correct triple $KL_{5,1}$, $KO_{5,2}$ and $KI_{5,2}$ is expected to be uniquely identified. This requires one round of KASUMI to be unwound for each choice of values for $KI_{5,1}$ and $KO_{5,2}$ and each of the eight differential pairs. Hence the work factor is equivalent to at most 1.6×2^{32} five round KASUMI encryptions.

Once the subkey $KL_{5,1}$ has been recovered, the sixteen rightmost bits h of the input difference to $FO5$ are known for each ciphertext pair. Therefore for each right pair the output difference $(k <<< 13) \oplus h$ of $FI51$ is known. Guessing the subkeys $KL_{5,2}$ and $KO_{5,1}$ allows the input values to $FI51$ to be computed. Hence, using only those ciphertext pairs that suggested the triple $KL_{5,1}$, $KO_{5,2}$ and $KI_{5,2}$, the triple $KL_{5,2}$, $KO_{5,1}$ and $KI_{5,1}$ can be recovered using a similar technique to that used to recover $KL_{5,1}$, $KO_{5,2}$ and $KI_{5,2}$.

The subkeys $KO_{5,3}$ and $KI_{5,3}$ are then easily recovered using the already recovered round key parts and the six ciphertext pairs whose output differences are not of the form $Y \oplus Y^* = cge$, where $g = e \oplus (k <<< 5)$. For each guess of $KO_{5,3}$, the input values to $FI53$ can be computed for each ciphertext pair. Since the output difference of $FI_{5,3}$ is also known for each right pair, a similar method to that described above can be used to recover the actual values of $KO_{5,3}$ and $KI_{5,3}$. The correct values of $KO_{5,3}$ and $KI_{5,3}$ are expected to be uniquely identified in at most 2^{16} trials. Note that the subkeys $KO_{5,3}$ and $KI_{5,3}$ can also be recovered by a brute force search in at most 2^{32} five round KASUMI encryptions.

Note that if all four right pairs have identical bit values in one or more common locations within the leftmost sixteen bits of the ciphertexts then there are at least two key pairs $KL_{5,1}$ and $KO_{5,2}$ giving the correct input values to $FI52$ for all four right pairs. A similar property applies to $KL_{5,2}$ and $KO_{5,1}$. Hence with probability ≈ 0.22 the correct key will not be uniquely identified. However, the attack is easily modified to cope with such an event, and the expected impact on running time is minimal.

This attack was implemented on a 400 Mhz Alpha and took approximately five hours to complete.

5 An Attack on Six Round KASUMI

This section describes a related key attack on six round KASUMI. The idea behind the attack is to obtain ciphertext pairs with a property such that it is possible to work back through round six without having to guess all of the round six round key bits, then use round five and the differential of Section 3 to suggest values for the missing key bits, and finally to use round four to help eliminate incorrect values for the key. The attack requires the encryption of

on average 3×2^{13} chosen plaintexts under the original key and 3×2^{17} chosen plaintexts under a chosen related key, and is expected to recover the entire key in a maximum of 2^{112} trials, equivalent to 1.5×2^{113} six round KASUMI encryptions.

When encrypting plaintexts X and X^* under keys K and K^* respectively, where X, X^*, K and K^* are as described in Section 3, with probability 2^{-16} the ciphertext difference is $Y \oplus Y^* = nge$, where n is any 32-bit string, g is any 16-bit string and $e = k <<< 8$. When $wt(k) = 1$ the attack requires an average of six ciphertext pairs of this form, and it is expected that these ciphertext pairs can be found from a set of 3×2^{17} ciphertext pairs generated from inputs of the appropriate form. One method of generating this set is to obtain 2^{15} chosen plaintext encryptions under the key K and for each such encryption obtain the sixteen related encryptions that result from using the different values of k such that $wt(k) = 1$. Note that with K and K^* as defined above, the only non zero round key part differences in rounds five and six are $\Delta KO_{5,3} = k <<< 13$ and $\Delta KO_{6,2} = k <<< 8$. Note also that the rightmost 32 bits of the plaintexts do not need to be known.

Note that for each of the six ciphertext pairs, the input difference to $FI62$ is 0^{16}, since $\Delta KO_{6,2} = k <<< 8 = e$. As $\Delta KI_{6,2} = 0^{16}$, the output difference of $FI62$ is also 0^{16}, so that there are only 2^{16} possible outputs of $FI62$ for each pair. Therefore, if the 16-bit output t of $FI62$ is known for any of these ciphertext pairs, it is not necessary to know the round keys $KO_{6,2}$ and $KI_{6,2}$ in order to compute the output of round six.

The attack guesses the ninety six key bits KL_6, $KO_{6,1}$, $KO_{6,3}$, $KI_{6,1}$ and $KI_{6,3}$ (so that the only unknown parts of the fifth round key are $KI_{5,1}$ and $KO_{5,3}$), and then takes each possible value of the string t in turn and assumes it to be the output of $FI62$ for all six ciphertext pairs. These values are then used along with the ciphertext values to compute the input values of $FI51$ and the output values of $FI52$ for all ciphertext pairs. The attack then assumes that all ciphertext pairs follow the differential of Section 3. This enables the output difference of $FI51$ to be computed for each ciphertext pair. The difference tables of $S7$ and $S9$ can then be used to suggest values for $KI_{5,1}$. Note that key bits must be suggested by both $S7$ and $S9$ in order for a value to be suggested for $KI_{5,1}$. On average one value of $KI_{5,1}$ will be suggested by each ciphertext pair.

Each value of $KI_{5,1}$ suggested by a ciphertext pair is then used to compute the two outputs of $FI51$ for that pair. These outputs are then XORed with the sixteen rightmost input bits to $FO5$ and the resulting values used along with the output difference $(k <<< 5) \oplus (k <<< 8) \oplus g$ of $FI53$ to suggest values for $KO_{5,3}$ by considering $FI53$ as a key dependent S-box, since the key $KI_{5,3}$ is equivalent to the already guessed key $KO_{6,3}$. On average one value of $KO_{5,3}$ will be suggested by each ciphertext pair.

If values are suggested for both $KI_{5,1}$ and $KO_{5,3}$ then each suggested pair $KI_{5,1}$, $KO_{5,3}$ is used along with the ninety six guessed key bits to compute from the two ciphertext values the output difference of $FL4$. If this output difference is equal to the input difference to $FL5$ then the pair $KI_{5,1}$, $KO_{5,3}$ is checked against a list of possible values for the key pair suggested already by

that ciphertext pair for different values of t but the same value of the ninety six key bits KL_6, $KO_{6,1}$, $KO_{6,3}$, $KI_{6,1}$ and $KI_{6,3}$ and, if not already there, added to the list. If the suggested key pair already appears in the list, then it should not be added again. This is to prevent the same key pair being suggested more than once by a given ciphertext pair and value of KL_6, $KO_{6,1}$, $KO_{6,3}$, $KI_{6,1}$ and $KI_{6,3}$. The probability of wrong keys generating the correct output difference of $FL4$ is 2^{-32}.

It is expected that three of the six ciphertext pairs follow the differential of Section 3. Therefore, taking the number of times a pair $KI_{5,1}$, $KO_{5,3}$ (equivalently $KO_{6,2}$, $KI_{6,2}$) is suggested for a particular choice of KL_6, $KO_{6,1}$, $KO_{6,3}$, $KI_{6,1}$ and $KI_{6,3}$ as the number of times the key KL_6, KO_6 and KI_6 is suggested, the correct value of the key KL_6, KO_6 and KI_6 is expected to be suggested at least three times. An incorrect key has probability 2^{-64} of being suggested by any one particular ciphertext pair and value of t, so that the probability of an incorrect key being suggested by any one particular pair is less than 2^{-48}. Therefore, the probability that an incorrect key is suggested by at least three out of the six pairs is less than 2^{-139}, and the probability that at least one incorrect key is suggested as many as three times is less than 2^{-11}. It is therefore expected that the correct key is the only key to be suggested as many as three times.

6 Conclusion

This paper describes related key attacks on both five and six round KASUMI. Under the assumption that it is possible to obtain the encryptions of approximately 2^{19} chosen plaintext pairs X and X^* under keys K and K^* respectively, where K and K^* differ in only one bit, the five round attack is practical in terms of computing power. The six round attack, while requiring fewer chosen plaintext encryptions than the five round attack, is of theoretical interest only due to the computing power required to implement the attack.

The attacks described above both rely on the same differential that predicts differences at the end of the third round with probability one half. This differential arises from the use of a very simple key schedule and the locations of where round key parts dependent upon K_3 are fed into the first three rounds. Changing the locations of where these round key parts are fed into the rounds may destroy this particular differential, but doesn't necessarily eliminate the possibility of constructing similar differentials.

Note that the attacks described in this paper present no real security threat to KASUMI as used in 3GPP as firstly they are only valid against reduced round variants of KASUMI, and secondly they require plaintexts to be encrypted under related keys in a way that should not be possible within the 3GPP environment.

References

1. S. Babbage, "Design of Security Algorithms for Third Generation Mobile Telephony." In *Information Security Technical Report (Elsevier)*, (5), 2000.

2. E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys." In *Advances in Cryptology - EUROCRYPT '93*, Lecture Notes in Computer Science (LNCS 765), Springer-Verlag, 1994.
3. E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems." In *Journal of Cryptology*, (4), 1991.
4. ETSI, see http://www.etsi.org/dvbandca/3GPP/3gppspecs.htm
5. 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*, Lecture Notes in Computer Science (LNCS 1109), Springer-Verlag, 1996.
6. L. Knudsen, "Cryptanalysis of LOKI91." In *Advances in Cryptology - AUSCRYPT '92*, Lecture Notes in Computer Science (LNCS 718), Springer-Verlag, 1994.
7. M. Matsui, "New Block Encryption Algorithm MISTY." In *Fast Software Encryption: 4th International Workshop*, Lecture Notes in Computer Science (LNCS 1267), Springer-Verlag, 1997.
8. K. Nyberg, "Linear Approximation of Block Ciphers." In *Advances in Cryptology - EUROCRYPT '94*, Lecture Notes in Computer Science (LNCS 950), Springer-Verlag, 1995.
9. K. Nyberg and L. Knudsen, "Provable Security Against a Differential Attack." In *Journal of Cryptology*, (8), 1995.

Security of Camellia against Truncated Differential Cryptanalysis

Masayuki Kanda[1,2] and Tsutomu Matsumoto[2]

[1] NTT Information Sharing Platform Laboratories
1-1 Hikari-no-oka, Yokosuka-shi, Kanagawa, 239-0847 Japan
kanda@isl.ntt.co.jp
[2] Yokohama National University
79-5 Tokiwadai, Hodogaya-ku, Yokohama-shi, Kanagawa, 240-8501 Japan
tsutomu@mlab.jks.ynu.ac.jp

Abstract. This paper studies security against truncated differential cryptanalysis from the "designer's" standpoint. In estimating the security, we use the upper bound of truncated differential probability. Previous works, Knudsen, Matsui and Moriai et al., searched for effective truncated differentials to attack byte-oriented block ciphers and computed the exact probability of the differentials. In this paper, we discuss the following items from the designer's standpoint; (a) truncated differential probability of effective active-s-box, (b) XOR cancellation probability, and (c) effect of auxiliary functions, e.g., FL/FL^{-1}-functions. We then combine them with Matsui's search algorithm and evaluate the security of Camellia, jointly developed by NTT and Mitsubishi Electric Corporation, against truncated differential cryptanalysis. We prove (from the designer's standpoint) that variants of Camellia with more than 11 rounds are secure against truncated differential cryptanalysis even if weak-key FL/FL^{-1}-functions are taken into consideration.

1 Introduction

At the second FSE in 1994, Knudsen proposed truncated differential cryptanalysis [K95], as an extension of differential cryptanalysis [BS93]. He defined truncated differentials as differentials where only a part of the difference in the ciphertexts (after a number of rounds) can be predicated. His first concept of truncated differentials was quite wide. Generally speaking, truncated differentials are now regarded as subsets of the characteristics that can be used to attack the cipher by using information on whether several bits of the difference are zero or not. In particular, "bytewise" truncated differentials, where one byte of the difference is regarded as 1 (non-zero) or 0 (zero), are useful in attacking byte-oriented block ciphers. For example, Knudsen and Berson attacked 5-round SAFER [KB96], and Matsui, Tokita [MT99,M99] and Moriai et al. [MSA+99] attacked the reduced version of E2 [KMA+00]. This means that we[1] have designed Camellia [AIK+00], jointly developed by NTT and Mitsubishi Electric

[1] Kanda is a member of the Camellia design team.

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 286–299, 2002.

Corporation, while considering its immunity to bytewise truncated differential cryptanalysis, since Camellia is a byte-oriented block cipher.

It is well known that there are two ways to estimate the security of ciphers against cryptanalyses; i.e., security measures from the "attacker's standpoint" and those from the "designer's standpoint." For example, in the case of differential cryptanalysis, one former measure is the maximum differential characteristic probability, and one of the latter is the upper bound of differential characteristic probability. Researchers who intend to attack ciphers have to find effective characteristic(s) and evaluate the exact characteristic probability from the attacker's standpoint in order to estimate the success rate of the attack and the computational load. On the other hand, it is sufficient for designers to show that the upper bound of the probability is low enough from the designer's standpoint in proving that the cipher is secure against differential cryptanalysis. This is because in such a case, all characteristics are useless in attacking the cipher, and it is unnecessary to calculate the stricter characteristic probability. Indeed, many recent ciphers have been designed on the basis of the upper bound of characteristic probability, such as SHARK [RDP+96], SQUARE [DKR97], Rijndael [DR98], SERPENT [ABK98], E2, Camellia, and MISTY [M97].

In this paper, we study security against truncated differential cryptanalysis from the designer's standpoint. Our evaluation is based on the upper bound of truncated differential probability.

First, we discuss the following items.

- How do effective active-s-boxes reduce truncated differential probability? Knudsen, Matsui, and Moriai et al. regarded 8-bit effective active-s-boxes as reducing the probability by 2^{-8}, but is this right (from the designer's viewpoint)?
- How do XOR cancellations reduce truncated differential probability? Matsui and Moriai et al. regarded them as reducing the probability by $(2^{-8})^h$, where h denotes the number of bytewise XOR cancellations. Is this right (from the designer's viewpoint)?
- How do auxiliary functions, e.g., FL/FL^{-1}-functions, revise truncated differential probability?

We then combine the above results with Matsui's search algorithm [M99] and evaluate the security of Camellia against truncated differential cryptanalysis.

This paper is organized as follows. In Sect. 2, we introduce some notations and definitions. The above items are discussed in Sect. 3 to Sect. 5, respectively. Finally, we show the security of Camellia against truncated differential cryptanalysis in Sect. 6, and conclude with a summary in Sect. 7.

2 Preliminaries

2.1 Notations

- $X = (x_1, \ldots, x_m),\ x_i \in \mathbb{Z}_2^n,\ (i = 1, \ldots, m)$: vector X over $\mathrm{GF}(2^n)^m$.
- $\Delta X = (\Delta x_1, \ldots, \Delta x_m),\ \Delta x_i \in \mathbb{Z}_2^n,\ (i = 1, \ldots, m)$: difference of X.

- $\delta X = (\delta x_1, \ldots, \delta x_m),\ \delta x_i \in \mathbb{Z}_2,\ (i = 1, \ldots, m)$: truncated differential of X.
- $\#\{S\}$: A number of elements in set S.

2.2 Definitions

Definition 1. *An active s-box is defined as an s-box given a non-zero input difference. When the s-box is bijective, it yields a non-zero output difference.*

Definition 2. *An Effective active-s-box is defined as an active s-box which outputs some specified difference; i.e., the output difference is related to the output difference(s) of other s-box(es). On the other hand, an active s-box which can output any non-zero difference is called a non-effective active-s-box.*

Definition 3. *When two non-zero differences are input to an XOR (exclusive-OR) operation but it outputs a zero difference, the XOR operation is called an XOR cancellation.*

Definition 4. *For any given Δx, $\Delta z \in \mathbb{Z}_2^n$, the differential probability of s-box $p_s(\Delta x \to \Delta z)$ is defined as:*

$$\begin{aligned} p_s(\Delta x \to \Delta z) &= \Pr_{x \in \mathbb{Z}_2^n}[s(x) \oplus s(x \oplus \Delta x) = \Delta z] \\ &= \frac{\#\{x \in \mathbb{Z}_2^n | s(x) \oplus s(x \oplus \Delta x) = \Delta z\}}{2^n} \end{aligned}$$

Definition 5. *For any given ΔX, $\Delta Y \in (\mathbb{Z}_2^n)^m$, the differential probability of F-function $p_F(\Delta X \to \Delta Y)$ is defined as:*

$$\begin{aligned} p_F(\Delta X \to \Delta Y) &= \Pr_{x \in (\mathbb{Z}_2^n)^m}[F(X) \oplus F(X \oplus \Delta X) = \Delta Y] \\ &= \frac{\#\{X \in (\mathbb{Z}_2^n)^m | F(X) \oplus F(X \oplus \Delta X) = \Delta Y\}}{(2^n)^m} \end{aligned}$$

Definition 6. *For any given $\Delta x \in \mathbb{Z}_2^n$, a function $\chi : \mathrm{GF}(2^n) \to \mathrm{GF}(2)$ is defined as:*

$$\chi(\Delta x) = \delta x = \begin{cases} 0 & \text{if } \Delta x = 0 \\ 1 & \text{if } \Delta x \neq 0 \end{cases}$$

Furthermore, for any given $\Delta X \in (\mathbb{Z}_2^n)^m$, $\chi(\Delta X) = \delta X = (\chi(\Delta x_1), \ldots, \chi(\Delta x_m))$.

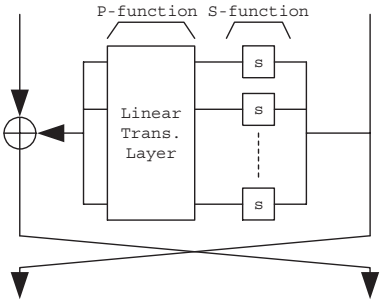


Fig. 1. Model

Definition 7. *For any given* $\Delta X, \Delta Y \in \mathrm{GF}(2^n)^m$ *and* $\delta X, \delta Y \in \mathrm{GF}(2)^m$*, the truncated differential probability of* F*-function* $p_F(\delta X \to \delta Y)$ *is defined as:*

$$p_F(\delta X \to \delta Y) = \frac{\sum\limits_{\substack{\chi(\Delta X) = \delta X \\ \chi(\Delta Y) = \delta Y}} \Pr\limits_{X \in (\mathbb{Z}_2^n)^m}[F(X) \oplus F(X \oplus \Delta X) = \Delta Y]}{\#\{\Delta X | \chi(\Delta X) = \delta X\}}$$

The following useful theorem can be obtained easily from Definition 7.

Theorem 1. *The upper bound of the truncated differential probability of* F*-function* $p_F(\delta X \to \delta Y)$ *is denoted as:*

$$\max_{\substack{\Delta X \\ \chi(\Delta X) = \delta X}} \sum_{\chi(\Delta Y) = \delta Y} \Pr_{X \in (\mathbb{Z}_2^n)^m}[F(X) \oplus F(X \oplus \Delta X) = \Delta Y]$$

Theorem 1 means that the upper bound of truncated differential probability $p_F(\delta X \to \delta Y)$ is obtained from the case that some input difference ΔX, which satisfies $\chi(\Delta X) = \delta X$, yields the maximum value of the above equation. Conversely, in order to prove immunity to truncated differential cryptanalysis, the designer should consider the upper bound of the probability assuming that difference ΔX which satisfies $\chi(\Delta X) = \delta X$ and yields the maximum value of the above equation is input to the F-function.

2.3 Model

Throughout this paper, we consider Feistel ciphers with SPN round function (See Fig. 1). A nonlinear layer that consists of m parallel s-boxes is called S-function and a linear transformation layer is called P-function. Hereafter, we assume that the s-boxes are 8-bit bijective substitution tables, since this eases the discussion.

Note that, since we assume that a round key, which is input to one round function, is generated independently and uniform randomly, and is bitwise XORed with input data, we ignore the effect of the round key throughout this paper.

3 Impact of Effective Active-s-Boxes

In this section, we study the effect of effective active-s-boxes.

As mentioned in Definition 1, an active s-box transforms a non-zero input difference to a non-zero output difference with some differential probability computed by Definition 4 because the s-box is bijective. Here, since the s-box is a nonlinear transformation function, the differential probability is *not* one in general. That is, $p_s(\Delta x \to \Delta z) < 1$ for $\Delta x \neq 0$ and $\Delta z \neq 0$. This means that an active s-box always reduces the probability of differential characteristics. Accordingly, one security measure against differential cryptanalysis is based on the minimum number of active s-boxes in the cipher, which shows the upper bound of differential characteristic probability.

A truncated differential just shows whether some bytewise difference is non-zero or zero, while the bytewise difference itself has some non-zero value. Thus, an active s-box that is bijective *always* transforms a non-zero bytewise input truncated differential to a non-zero bytewise output truncated differential; i.e., $p_s(\delta x \to \delta z) = 1$ for $\delta x \neq 0$ and $\delta z \neq 0$. Such an active s-box is a non-effective active-s-box, see Definition 2, since it does *not* reduce the probability of truncated differentials. Accordingly, when each active s-box outputs a non-zero output difference independently, all active s-boxes are non-effective active-s-boxes and do not reduce any truncated differential probability.

On the other hand, how do active s-boxes reduce the truncated differential probability if they output some dependent output difference(s)? Here, a dependent output difference means the output difference that is determined by output difference(s) of other active s-box(es). In this case, it is necessary to consider the output difference(s) itself of the active s-box(es) and not the output truncated differential(s). Such an active s-box is an effective active-s-box mentioned in Definition 2, and it reduces the probability of truncated differentials in the same way as an active s-box in differential cryptanalysis.

Hereafter, we show how effective active-s-boxes reduce the truncated differential probability.

It is well known that the inverse function in $\mathrm{GF}(2^8)$ provides the minimum value of the maximum differential probability, i.e., 2^{-6}. Thus, many ciphers use s-boxes based on the inverse function in $\mathrm{GF}(2^8)$. Here, assume that all s-boxes consist of combinations of the inverse function in $\mathrm{GF}(2^8)$ and an affine transformation. $\delta X, \delta Z$ denote an input truncated differential and an output truncated differential of S-function, respectively.

Theorem 2. *Assume that all s-boxes consist of combinations of the inverse function in* $\mathrm{GF}(2^8)$ *and an affine transformation. Let the number of active s-boxes be k ($k \geq 2$). When all active s-boxes output the same output difference*[2]*, the truncated differential probability of S-function $p_S(\delta X \to \delta Z)$ satisfies the following relationship.*

$$p_S(\delta X \to \delta Z) \leq 2^{-7(k-1)}(1 + 2^{k-7} - 2^{-6})$$

[2] In this case, the number of effective active-s-boxes is $k - 1$.

Proof. It is easily confirmed that the difference distribution table of the inverse function in GF(2^8) has one 4-possible pairs and 126 2-possible pairs in each column and each row. Furthermore, an affine transformation only interchanges the columns of the difference distribution table. Thus, for each Δz, the differential probability of s-box $p_s(\Delta x \to \Delta z)$ is 2^{-6} for one entry and 2^{-7} for 126 entries from 256 entries of Δx, since the s-box consists of combinations of the inverse function in GF(2^8) and an affine transformation.

Accordingly, in order to calculate the upper bounds of $p_S(\delta X \to \delta Z)$ when all active s-boxes output the same output difference Δz, we should consider that the same input difference Δx is input to all s-boxes. This leads to the following relationship.

$$p_S(\delta X \to \delta Z) \leq \max_{\Delta x \neq 0} \sum_{\Delta z \neq 0} \{p_s(\Delta x \to \Delta z)\}^k = (2^{-6})^k + 126 \cdot (2^{-7})^k$$

Q.E.D.

Theorem 2 can easily be extend for the generic case as follows.

Theorem 3. *Assume that each row (Δx) of the difference distribution table of an 8-bit s-box has n_i 2^{8-i}-possible pairs ($i = 0, 1, \ldots, 7$)[3]. Let the number of active s-boxes be k ($k \geq 2$). When all active s-boxes output the same output difference[4], the truncated differential probability of S-function $p_S(\delta X \to \delta Z)$ satisfies the following relationship.*

$$p_S(\delta X \to \delta Z) \leq \max_{\Delta x \neq 0} \left\{ 2^{-7(k-1)} + \sum_{i=0}^{6} \frac{n_i}{2^i} \left(2^{-i(k-1)} - 2^{-7(k-1)} \right) \right\}$$

Proof. This theorem can be obtained easily from the following two equations.

$$\sum_{\Delta z \neq 0} \{p_s(\Delta x \to \Delta z)\}^k = \sum_{i=0}^{7} n_i \cdot 2^{-ik}, \quad \sum_{i=0}^{7} n_i \cdot 2^{8-i} = 256$$

Q.E.D.

Theorem 4. *Assume that all s-boxes consist of combinations of the inverse function in* GF(2^8) *and an affine transformation. Let the number of active s-boxes be 3. Assume that an active s-box outputs the output difference denoted by the XOR of two other output differences[5]. That is, these output differences represent $(\alpha, \beta, \alpha \oplus \beta), \alpha \neq \beta$. Then, the truncated differential probability of S-function $p_S(\delta X \to \delta Z)$ satisfies the following relationship.*

$$p_S(\delta X \to \delta Z) \leq 2^{-7} + 2^{-14} + 2^{-20} \simeq 2^{-7}$$

[3] In the case of Theorem 2, $n_0 = n_1 = n_2 = n_3 = n_4 = n_5 = 0, n_6 = 1, n_7 = 126$.
[4] In this case, the number of effective active-s-boxes is $k - 1$.
[5] In this case, the number of effective active-s-boxes is 1.

Proof. As mentioned at the proof of Theorem 2, for each Δz, the differential probability of s-box $p_s(\Delta x \to \Delta z)$ is 2^{-6} for one entry and 2^{-7} for 126 entries from 256 entries of Δx. Also, for each Δx, the differential probability is 2^{-6} for one entry and 2^{-7} for 126 entries from 256 entries of Δz.

Let $\Delta x_1, \Delta x_2, \Delta x_3$ denote input differences to active s-boxes. Without loss of generality, we assume that $p_s(\Delta x_1 \to \alpha_0) = 2^{-6}$, $p_s(\Delta x_2 \to \beta_0) = 2^{-6}$, $p_s(\Delta x_3 \to \alpha_0 \oplus \beta_0) = 2^{-6}$, and $\alpha_0 \neq \beta_0$. The differential probability and the entries of such probability are determined as follows by combinations of the output differences $(\alpha, \beta, \alpha \oplus \beta)$.

1. $(\Delta x_1, \Delta x_2, \Delta x_3) \mapsto (\alpha_0, \beta_0, \alpha_0 \oplus \beta_0)$: the differential probability is $(2^{-6})^3$ and the entry is 1.
2. $(\Delta x_1, \Delta x_2, \Delta x_3) \mapsto (\alpha_0, \beta, \alpha_0 \oplus \beta),\ \beta \neq \beta_0$: the differential probability is $2^{-6} \cdot (2^{-7})^2$ and the entries are (at most) 126.
3. $(\Delta x_1, \Delta x_2, \Delta x_3) \mapsto (\alpha, \beta_0, \alpha \oplus \beta_0),\ \alpha \neq \alpha_0$: the differential probability is $2^{-6} \cdot (2^{-7})^2$ and the entries are (at most) 126.
4. $(\Delta x_1, \Delta x_2, \Delta x_3) \mapsto (\alpha, \alpha \oplus \alpha_0 \oplus \beta_0, \alpha_0 \oplus \beta_0),\ \alpha \neq \alpha_0$: the differential probability is $2^{-6} \cdot (2^{-7})^2$ and the entries are (at most) 126.
5. $(\Delta x_1, \Delta x_2, \Delta x_3) \mapsto (\alpha, \beta, \alpha \oplus \beta),\ \alpha \neq \alpha_0,\ \beta \neq \beta_0,\ \alpha \oplus \beta \neq \alpha_0 \oplus \beta_0$: the differential probability is $(2^{-7})^3$ and the entries are (at most) $126 \cdot 125$.

From the above, Theorem 1 leads to the following relationship.

$$\begin{aligned} p_S(\delta X \to \delta Z) &\leq \max_{\substack{(\Delta x_1, \Delta x_2, \Delta x_3) \\ \chi(\Delta X) = \delta X}} \sum_{\substack{(\alpha_0, \beta_0), (\alpha_0, \beta), \\ (\alpha, \beta_0), (\alpha, \beta)}} p_S(\Delta X \to \Delta Z) \\ &= 2^{-18} + 126 \cdot 2^{-20} + 126 \cdot 2^{-20} + 126 \cdot 2^{-20} + 126 \cdot 125 \cdot 2^{-21} \\ &= 2^{-7} + 2^{-14} + 2^{-20} \end{aligned}$$

Q.E.D.

Theorem 5. *Assume that all s-boxes consist of combinations of the inverse function in* $\mathrm{GF}(2^8)$ *and an affine transformation. Let the number of active s-boxes be 7. Assume that seven output differences represent* $(\alpha, \alpha, \alpha, \beta, \beta, \alpha \oplus \beta, \alpha \oplus \beta)$[6]*. Then, the truncated differential probability of S-function* $p_S(\delta X \to \delta Z)$ *satisfies the following relationship.*

$$p_S(\delta X \to \delta Z) \leq 2^{-35} + 2^{-38} - 2^{-40} - 2^{-44} + 3 \cdot 2^{-48} \simeq (2^{-7})^5$$

Since the proof of this theorem is very similar to the proof of Theorem 4, the proof is omitted here. From Theorem 2, Theorem 4, and Theorem 5, we can obtain the following conjecture.

Conjecture 1. *Assume that all s-boxes consist of combinations of the inverse function in* $\mathrm{GF}(2^8)$ *and an affine transformation. The upper bound of truncated differential probability is reduced by approximately* 2^{-7} *per effective active-s-box*[7]*.*

[6] In this case, the number of effective active-s-boxes is 5.

[7] It is expected that the stricter upper bound is $2^{-6.9}$.

Knudsen, Matsui and Moriai et al. estimated that the truncated differential probability is reduced by approximately 2^{-8} (exactly $\frac{1}{255}$) per effective active-s-box assuming that the output difference distribution of the s-box is uniform. However, since the distribution of an even-bit s-box is not uniform, designers should not ignore the case when truncated differential probability is reduced by approximately 2^{-7} per effective active-s-box. Accordingly, we regard an effective active-s-box as reducing the probability by 2^{-7} to evaluate the security of Camellia against truncated differential cryptanalysis.

4 XOR Cancellation Probability

Let $\Delta X^{(r-1)}$, $\delta X^{(r-1)}$ be an input difference and an input truncated differential in the $(r-1)$-th round, respectively. Let $\Delta Y^{(r)}$, $\delta Y^{(r)}$ be an output difference and an output truncated differential in the r-th round, respectively. Here, $(\Delta X^{(0)}, \Delta X^{(1)})$ denotes a difference of plaintexts.

$$\Delta X^{(r-1)} = (\Delta x_1^{(r-1)}, \ldots, \Delta x_m^{(r-1)}) \ , \ \ \delta X^{(r-1)} = (\delta x_1^{(r-1)}, \ldots, \delta x_m^{(r-1)})$$
$$\Delta Y^{(r)} = (\Delta y_1^{(r)}, \ldots, \Delta y_m^{(r)}) \ , \ \ \delta Y^{(r)} = (\delta y_1^{(r)}, \ldots, \delta y_m^{(r)})$$

An input difference $\Delta X^{(r+1)}$ and an input truncated differential $\delta X^{(r+1)}$ in the $(r+1)$-th round are obtained as follows.

$$\Delta X^{(r+1)} = (\Delta x_1^{(r-1)} \oplus \Delta y_1^{(r)}, \ldots, \Delta x_m^{(r-1)} \oplus \Delta y_m^{(r)})$$
$$\delta X^{(r+1)} = (\delta x_1^{(r-1)} \oplus \delta y_1^{(r)}, \ldots, \delta x_m^{(r-1)} \oplus \delta y_m^{(r)})$$

Since an XOR cancellation occurs when $\Delta x_i^{(r-1)} = \Delta y_i^{(r)} (\neq 0)$, XOR cancellation probability (of byte operation) is approximately 2^{-8} (exactly $\frac{1}{255}$) if $\Delta x_i^{(r-1)}$ and/or $\Delta y_i^{(r)}$ is determined uniform randomly. Thus, in the previous estimation, the XOR cancellation probability of truncated differentials denotes $(2^{-8})^h$, where h is the number of XOR cancellations. Here, $h = hw(\delta X^{(r-1)} \vee \delta Y^{(r)}) - hw(\delta X^{(r+1)})$ $(\leq hw(\delta X^{(r-1)} \wedge \delta Y^{(r)}))$, and $hw(X)$ denotes the Hamming weight of X.

Note that, when the XOR cancellation probability is satisfied as mentioned above, there is a tacit assumption that no relation exists between $\Delta x_i^{(r-1)}$ and $\Delta x_j^{(r-1)}$ $(i \neq j)$ or between $\Delta y_i^{(r)}$ and $\Delta y_j^{(r)}$ $(i \neq j)$. Generally speaking, it is expected that there is no relation between $\Delta x_i^{(r-1)}$ and $\Delta x_j^{(r-1)}$ $(i \neq j)$ if $(r-1)$ is large. However, this cannot be expected if $(r-1)$ is small. Accordingly, XOR cancellation probability is dependent on whether there is a relation between $\Delta y_i^{(r)}$ and $\Delta y_j^{(r)}$ $(i \neq j)$ or not.

For example, consider E2 and Camellia as shown in Fig. 2. Since an S-P-S structure is applied to the round function of E2, output difference Δy_i is represented as follows.

$$\Delta y_i = s(f_i(s(\Delta x_1), \ldots, s(\Delta x_8))), \ (i = 1, \ldots, 8),$$

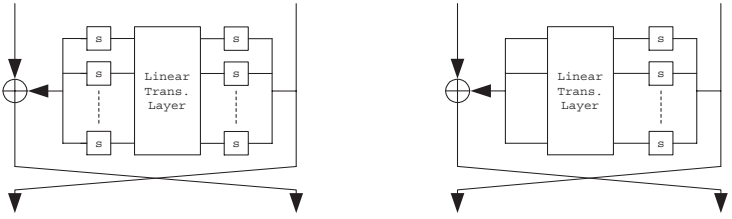


Fig. 2. Round Functions of E2 (left hand) and Camellia (right hand)

where $s()$ denotes an s-box and $f_i()$ shows a linear transformation. It turns out that, for any input difference ΔX belonging to the given input truncated differential δX, there is no relation between Δy_i and Δy_j $(i \neq j)$ even if $f_i(s(\Delta x_1), \ldots, s(\Delta x_8)) = f_j(s(\Delta x_1), \ldots, s(\Delta x_8))$ for some i, j. That is, all output differences Δy_i are determined independently. Accordingly, as in the previous estimation, the XOR cancellation probability of E2 can be represented as $(2^{-8})^h$, where h denotes the number of bytewise XOR cancellations.

On the other hand, since an S-P structure is applied to the round function of Camellia, output difference Δy_i is represented as follows.

$$\Delta y_i = f_i(s(\Delta x_1), \ldots, s(\Delta x_8)),\ (i = 1, \ldots, 8).$$

More concretely, we can rewrite as follows.

$$\begin{aligned}
\Delta y_1 &= s(\Delta x_1) \oplus s(\Delta x_3) \oplus s(\Delta x_4) \oplus s(\Delta x_6) \oplus s(\Delta x_7) \oplus s(\Delta x_8)\\
\Delta y_2 &= s(\Delta x_1) \oplus s(\Delta x_2) \oplus s(\Delta x_4) \oplus s(\Delta x_5) \oplus s(\Delta x_7) \oplus s(\Delta x_8)\\
\Delta y_3 &= s(\Delta x_1) \oplus s(\Delta x_2) \oplus s(\Delta x_3) \oplus s(\Delta x_5) \oplus s(\Delta x_6) \oplus s(\Delta x_8)\\
\Delta y_4 &= s(\Delta x_2) \oplus s(\Delta x_3) \oplus s(\Delta x_4) \oplus s(\Delta x_5) \oplus s(\Delta x_6) \oplus s(\Delta x_7)\\
\Delta y_5 &= s(\Delta x_1) \oplus s(\Delta x_2) \oplus s(\Delta x_6) \oplus s(\Delta x_7) \oplus s(\Delta x_8)\\
\Delta y_6 &= s(\Delta x_2) \oplus s(\Delta x_3) \oplus s(\Delta x_5) \oplus s(\Delta x_7) \oplus s(\Delta x_8)\\
\Delta y_7 &= s(\Delta x_3) \oplus s(\Delta x_4) \oplus s(\Delta x_5) \oplus s(\Delta x_6) \oplus s(\Delta x_8)\\
\Delta y_8 &= s(\Delta x_1) \oplus s(\Delta x_4) \oplus s(\Delta x_5) \oplus s(\Delta x_6) \oplus s(\Delta x_7)
\end{aligned}$$

Now, let an input truncated differential be $\delta X = (10001000)$. The above equations are represented as;

$$\begin{aligned}
&\Delta y_1 = \Delta y_5 = s(\Delta x_1),\\
&\Delta y_2 = \Delta y_3 = \Delta y_8 = s(\Delta x_1) \oplus s(\Delta x_5),\\
&\Delta y_4 = \Delta y_6 = \Delta y_7 = s(\Delta x_5).
\end{aligned}$$

This means that the relation between Δy_1 and Δy_5 is always satisfied regardless of input difference Δx_1. so also are the relations between Δy_2, Δy_3 and Δy_8, and between Δy_4, Δy_6 and Δy_7.

Let $\delta X^{(r)} = (10001000)$, $\delta X^{(r-1)} = (01100001)$, and $\Delta x_2^{(r-1)} = \Delta x_3^{(r-1)} = \Delta x_8^{(r-1)} = \alpha$, for example. If an input truncated differential of the $(r+1)$-th

round satisfies $\delta X^{(r+1)} = (10011110)$, then the XOR cancellation probability is 2^{-8}, not $(2^{-8})^3$, since it is sufficient to satisfy $s(\Delta x_1) \oplus s(\Delta x_5) = \alpha$. Accordingly, we obtain the following theorem.

Theorem 6. *Let input truncated differentials of the $(r-1)$-th round and the $(r+1)$-th round be $\delta X^{(r-1)}$ and $\delta X^{(r+1)}$, respectively. Also, let an output truncated differential and an output difference of the r-th round be $\delta Y^{(r)}$ and $\Delta Y^{(r)}$, respectively. Furthermore, an (i,j) element of $m \times m$ matrix $\mathcal{D}$ is determined by;*

$$d_{ii} = \begin{cases} 1 & \text{if} \quad \delta x_i^{(r-1)} \cdot \delta y_i^{(r)} = 1 \cap \delta x_i^{(r+1)} = 0 \\ 0 & \text{otherwise} \end{cases}$$
$$d_{ij} = 0 \qquad \text{if} \quad i \neq j$$

$\mathcal{D} \cdot \Delta Y^{(r)}$ picks up the value(s) which leads to XOR cancellation. The same value of $\mathcal{D} \cdot \Delta Y^{(r)}$ means that there is a relation between the output differences. Accordingly, XOR cancellation probability is represented as approximately $(2^{-8})^k$, where k is the rank of $\mathcal{D} \cdot \Delta Y^{(r)}$. Note that, since S-P-S round function always makes the rank $k = h$, the previous estimation is a special case of this theorem.

5 Impact of Auxiliary Functions

Auxiliary functions, which are heterogeneous functions from round function (core structure), are inserted before and after the core and/or at the inside of the core. Generally speaking, it is expected that auxiliary functions improve the security. Thus, if the core has sufficient immunity to cryptanalyses, it is considered that the cipher, which has the core and auxiliary functions, is also secure. However, if an auxiliary function is a key-dependent linear transformation such as the FL/FL^{-1}-function of Camellia, there is some possibility of generating weak-key auxiliary functions. That is, whether an auxiliary function decreases the security depends on the key used. This means that designers should estimate the security of the cipher assuming the use of weak-key auxiliary functions.

In this section, we study an effect on security when inserting key-dependent linear transformation functions as auxiliary functions. Here, we focus on the FL-function of Camellia. The differential transition of the FL-function is described as follows.

$$\begin{aligned} FL : \mathrm{GF}(2^{32})^4 &\to \mathrm{GF}(2^{32})^2; \\ (\Delta A, \Delta B, k_1, k_2) &\mapsto (\Delta C, \Delta D), \\ \Delta D &= ((\Delta A \wedge k_1) <<< 1) \oplus \Delta B), \\ \Delta C &= \Delta A \oplus (\Delta D \wedge \overline{k_2}), \end{aligned}$$

where $<<< 1$ denotes one-bit right circular rotation, k_1, k_2 are subkeys, and $\overline{k_2}$ represents the complement of k_2. Thus, through the FL-function, an input difference of the FL-function $(\Delta A, \Delta B)$ is mapped to some output difference $(\Delta C, \Delta D)$ by;

$$\begin{aligned} (\Delta A, \Delta B) = (0,0) &\overset{k_1,k_2}{\longmapsto} (\Delta C, \Delta D) = (0,0), \\ (\Delta A, \Delta B) \neq (0,0) &\overset{k_1,k_2}{\longmapsto} (\Delta C, \Delta D) \in \mathrm{GF}(2^{32})^2 \backslash (0,0). \end{aligned}$$

Since $(\Delta A, \Delta B) \neq (0,0)$ is transformed to some $(\Delta C, \Delta D) \in \mathrm{GF}(2^{32})^2 \backslash (0,0)$, there is some possibility of transforming $(\Delta A', \Delta B')$ with relatively high differential probability to $(\Delta C, \Delta D) \neq (0,0)$, even if $(\Delta C, \Delta D) \neq (0,0)$ has low differential probability before applying a weak-key FL-function. This means that taking the weak-key FL-function into consideration is equivalent to regarding the probability of all non-zero output differences of the FL-function as the maximum probability of non-zero input difference of the FL-function.

The above statement is applicable to the FL^{-1}-function of Camellia. Accordingly, we define the effect of FL/FL^{-1}-function as follows.

Definition 8. *Let δI, δO be an input truncated differential and an output truncated differential of FL/FL^{-1}-function, respectively. The effect of FL/FL^{-1}-function is that the probability of non-zero truncated differentials is changed to;*

$$\forall \delta O \neq 0, \quad \Pr((\delta X(0), \delta X(1)) \to \delta O) = \max_{\delta I \neq 0} \Pr((\delta X(0), \delta X(1)) \to \delta I),$$

where $(\delta X(0), \delta X(1))$ is a truncated differential of plaintexts.

6 Upper Bound of Best Bytewise Characteristics of Camellia

In this section, we look for the upper bound of best bytewise characteristics of Camellia. Our search algorithm is based on Matsui's search algorithm (width first basis) [M99], and we modify his algorithm at the following points.

Step 1: Generation of Truncated Differential Probability of *F*-Function

In [M99], since he assumed that a difference distribution table of s-box was uniform, he regarded the truncated differential probability as reducing 2^{-8} per effective active-s-box. As mentioned in Sect. 3, however, the difference distribution table of the s-box which consists of combinations of the inverse function in $\mathrm{GF}(2^8)$ and an affine transformation is not uniform, and the upper bound of truncated differential probability is reduced by approximately 2^{-7} per effective active-s-box. Thus, we regard the upper bound of truncated differential probability of F-function as being 2^{-7r}, where r is the number of effective active-s-boxes of F-function. r is obtained as follows for given input truncated differential δX and output truncated differential δY.

$$r = rank(\mathcal{F}(\overline{\delta Y}) P \delta X),$$

where $\overline{\delta Y}$ is the complement of δY, P denotes the matrix of the linear transformation in $\mathrm{GF}(2)$, and $\mathcal{F}(\overline{\delta Y})$ represents the 8×8 diagonal matrix whose (i,i) component equals $\overline{\delta y_i}$ for $i = 1, \ldots, 8$.

Step 2: Generation of Truncated Differential Probability of Round Function

In [M99], since he assumed that there was no relation among bytewise output differences of F-function, he regarded the XOR cancellation probability as being 2^{-8n}, where n is the number of XOR cancellation. As mentioned in Sect. 4, however, there are some relations among bytewise output differences of F-function, since the F-function of Camellia consists of S-P structure. Thus, Theorem 6 makes XOR cancellation probability 2^{-8k}, where k is the rank of $\mathcal{D} \cdot \Delta Y$.

The simple calculation on the rank of $\mathcal{D} \cdot \Delta Y$ is shown below.

Definition 9. *Let A, B, C and D denote 4-bit independent difference patterns, i.e, $A = 0001, B = 0010, C = 0100$, and $D = 1000$. Furthermore, assume that a given output difference $\Delta Z = (\Delta z_1, \ldots, \Delta z_8)$ can be labeled by combinations of the difference patterns A, B, C, and D. Rank representation of ΔZ is defined as the description of how each $Delta z_i (\neq 0)$, $(i = 1, \ldots, 8)$ is represented by a combination of A, B, C, and D.*

For example, let a difference ΔZ be $\Delta Z = (\Delta z_1, \Delta z_2, 0, 0, \Delta z_5, 0, \Delta z_7, \Delta z_8)$, and the three following relations exist; $\Delta z_1 = \Delta z_5$, $\Delta z_2 = \Delta z_8$, $\Delta z_7 = \Delta z_1 \oplus \Delta z_2$. When difference patterns are labeled as $A = \Delta z_1$ and $B = \Delta z_2$, the rank representation of ΔZ is $(A, B, 0, 0, A, 0, A \oplus B, B)$. Strict speaking, it is $(0001, 0010, 0000, 0000, 0001, 0000, 0011, 0010)$.

Algorithm 1 *(Simple calculation method on the rank of $\mathcal{D} \cdot \Delta Y$)*

S1 *Make a rank representation of an output difference of S-function ΔZ for a given input truncated differential δX.*

S2 *Transform the rank representation of ΔZ to a rank representation of output difference of F-function ΔY by P-function.*

S3 *Hold the rank representation of ΔY that coincides with given output truncated differential δY as a candidate of ΔY. If there are several candidates, do the following checks.*
- *Discard all but one candidate, when their rank representations are the same*[8]*.*
- *Keep the candidates whose rank is minimum.*
- *Keep the candidates whose number of the same difference patterns in the rank representation*[9] *is maximum, if there are several candidates whose rank is the same.*
- *Hold all the candidates that survive the above checks.*

S4 *Repeat S2 and S3 for other rank representations of ΔZ. After finishing the above process for all rank representations, all the candidates are regarded as (candidates of) ΔY.*

[8] The meaning of the same rank representations is that there is the same relations between output differences among the candidates; e.g., $(A, A, B, A \oplus B)$ and $(B, B, A \oplus B, A)$.

[9] For example, the numbers of the same difference patterns in $(A, A, B, A \oplus B, A \oplus B)$ and (A, B, B, B, A) are 2 and 3, respectively.

Table 1. Upper Bound of Bytewise Characteristic Probability of Camellia (log_2 representation)

Rounds	1	2	3	4	5	6	7	8	9	10	11
without FL/FL^{-1}-func.	0	0	-7	-21	-36	-51	-73	-93	—	—	—
with FL/FL^{-1}-func.	0	0	-7	-21	-36	-51	-51	-59	-59	-103	—

S5 *Generate an* 8×8 *matrix* $\mathcal{D}$ *by following Theorem 6, and calculate the rank of* $\mathcal{D} \cdot \Delta Y$ *for every* ΔY.

Step 3: Search Truncated Differential Characteristics of N Rounds

There is basically no modification of Matsui's search algorithm. The one modification is that, if FL/FL^{-1} functions are inserted, truncated differential probability is changed by following Definition 8 whenever truncated differentials pass through FL/FL^{-1} functions.

The search algorithm is terminated when (the upper bound of) truncated differentials become indistinguishable from random permutations. That is, the final round is the r-th round whose (upper bound of) maximum truncated differential probability is lower than $2^{-128+8h}$, where h is the Hamming weight of the truncated differential $(\delta X^{(r)}, \delta X^{(r+1)})$; i.e., $h = hw(\delta X^{(r)}) + hw(\delta X^{(r+1)})$. Note that, a random permutation outputs the same form of truncated differential $(\delta X^{(r)}, \delta X^{(r+1)})$ with probability $2^{-128+8h}$.

As shown in Table 1, we obtained the upper bound of best bytewise characteristic probability using the modified search algorithm. Table 1 shows that, if the number of rounds is more than 11, (reduced versions of) Camellia with FL/FL^{-1} functions is indistinguishable from random permutations in terms of truncated differential cryptanalysis. An interested thing is that Table 1 doesn't increase from the eighth round to the ninth round if FL/FL^{-1}-functions are present. This is because the input truncated differential of the ninth round becomes zero by XOR cancellation at the eighth round with probability 2^{-8}.

Thus, it is proven (from the designer's standpoint) that variants of Camellia with more than 11 rounds are secure against truncated differential cryptanalysis even if weak-key FL/FL^{-1} functions are taken into consideration. Note that, since the probability shown in Table 1 is the upper bound, we believe that Camellia is more secure than our estimation indicates.

7 Conclusion

In this paper, we studied security against truncated differential cryptanalysis from the "designer's" standpoint. Our evaluation is based on the upper bound of truncated differential probability, and we evaluated the security of Camellia against truncated differential cryptanalysis by using a modified version of Matsui's search algorithm.

We proved (from the designer's standpoint) that variants of Camellia with more than 11 rounds are secure against truncated differential cryptanalysis even if weak-key FL/FL^{-1} functions are taken into consideration. Note that, our result does NOT mean that 10-round variant of Camellia is breakable by truncated differential cryptanalysis, since the probability shown in Table 1 is the upper bound. That is, we did not show whether such effective truncated differentials exist.

References

[AIK+00] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, "Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms," *Selected Areas in Cryptography — 7th Annual International Workshop, SAC2000*, pp.41–54 in preproceeding, 2000, (LNCS to appear).

[BS93] E. Biham and A. Shamir, "Differential Cryptanalysis of the Data Encryption Standard," Springer-Verlag, Berlin, 1993.

[ABK98] R. Anderson, E. Biham, and L. R. Knudsen, "SERPENT," *The First Advanced Encryption Standard Candidate Conference*, 1998.
`http//www.cl.cam.ac.uk/~rja14/serpent.html`

[DKR97] J. Daemen, L. Knudsen, and V. Rijmen, "The block cipher SQUARE," *Fast Software Encryption — 4th International Workshop, FSE'97*, LNCS **1267**, pp.54–68, Springer-Verlag, Berlin, 1997.

[DR98] J. Daemen and V. Rijmen, "RIJNDAEL," *The First Advanced Encryption Standard Candidate Conference*, 1998.
`http//www.esat. kuleuven.ac.be/~rijmen/rijndael/`

[K95] L. R. Knudesen, "Truncated and Higher Order Differentials," *Fast Software Encryption — Second International Workshop*, LNCS **1008**, pp.196–211, Springer-Verlag, Berlin, 1995.

[KB96] L. R. Knudsen and T. A. Berson, "Truncated Differentials of SAFER," *Fast Software Encryption — Third International Workshop*, LNCS **1039**, pp.15–26, Springer-Verlag, Berlin, 1996.

[KMA+00] M. Kanda, S. Moriai, K. Aoki, H. Ueda, Y. Takashima, K. Ohta, and T. Matsumoto, "*E2* – A New 128-Bit Block Cipher," *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E83-A, No. 1, pp.48–59, 2000.

[M97] M. Matsui, "New Block Encryption Algorithm MISTY," *Fast Software Encryption — 4th International Workshop, FSE'97*, LNCS **1267**, pp.54–68, Springer-Verlag, Berlin, 1997.

[M99] M. Matsui, "Differential Path Search of the Block Cipher E2," *IEICE Technical report*, ISEC99-19, 1999-07. (In Japanese.)

[MT99] M. Matsui and T. Tokita, "Cryptanalysis of a Reduced Version of the Block Cipher E2," *Fast Software Encryption — 6th International Workshop, FSE'99*, LNCS **1636**, pp.71–80, Springer-Verlag, Berlin, 1999.

[MSA+99] S. Moriai, M. Sugita, K. Aoki, and M. Kanda, "Security of E2 against Truncated Differential Cryptanalysis," *Selected Areas in Cryptography — 6th Annual International Workshop, SAC'99*, LNCS **1758**, pp.106–117, Springer-Verlag, Berlin, 2000.

[RDP+96] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E.D. Win, "The cipher SHARK," *Fast Software Encryption — Third International Workshop*, LNCS **1039**, pp.99–111, Springer-Verlag, Berlin, 1996.

Impossible Differential Cryptanalysis of Zodiac

Deukjo Hong[1], Jaechul Sung[1], Shiho Moriai[2], Sangjin Lee[1], and Jongin Lim[1⋆]

[1] Center for Information Security Technologies(CIST),
Korea University, Anam Dong, Sungbuk Gu,
Seoul, Korea
{hongdj, sjames, sangjin, jilim}@cist.korea.ac.kr
[2] NTT Laboratories, 1-1 Hikarinooka, Yokosuka, 239-0847 Japan
shiho@isl.ntt.co.jp

Abstract. We discuss the impossible differential cryptanalysis of the block cipher Zodiac [7]. The main design principles of Zodiac are simplicity and efficiency. However the diffusion layer in its round function is too simple to offer enough security. An impossible differential cryptanalysis is a proper method to attack the weakness of Zodiac. Our attack using two 14-round impossible characteristics derives 128-bit master key of the full 16-round Zodiac with its complexity 2^{119} encryption times faster than the exhaustive search. The efficiency of the attack compared with exhaustive search increases as the key size increases.

1 Introduction

Differential cryptanalysis which was proposed by E. Biham and A. Shamir [3] is the most powerful attack for block ciphers. Later, it was regarded as a very useful method in attacking the known block ciphers – FEAL [10], LOKI [4], and so on. For these reasons, block ciphers have been designed to consider the differential cryptanalysis since the middle of 1990's. Differential cryptanalysis has also been advanced variously – Conditional Differential Cryptanalysis [1, 9], Truncated Differential Cryptanalysis [5], Impossible Differential Cryptanalysis [2], Higher Order Differential Cryptanalysis [5,6,8], Boomerang attack [11], and so on.

The conventional differential cryptanalysis finds a key using the differential characteristic with a high probability. The attacker chooses ciphertext pairs with a specific difference of plaintexts, discards wrong pairs by filtering, and then finds a key by applying the counting methods to the remaining pairs.

If a filtering method is efficient, the signal to noise ratio is greater than 1. However, the case that the signal to noise ratio is far less than 1 is also useful. Especially, the differential characteristic whose probability is zero is efficiently applied to attack block ciphers. This attack is called the impossible differential cryptanalysis.

⋆ This work is supported in part by the Ministry of Information & Communication of Korea ("Support Project of University Information Technology Research Center" supervised by IITA)

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 300–311, 2002.

In general, it is possible that a cipher which has a cryptographically weak diffusion or permutation has a long impossible differential characteristic, . E. Biham et al. found a 24-round impossible differential characteristic to analyze 31-round Skipjack in [2].

The general impossible differential cryptanalysis is as follows:

1. Find impossible differential characteristics.
2. Obtain the ciphertext pairs for their plaintext pairs with the input difference of the impossible differential characteristic.
3. For each value in the key space of the final round, decrypt the ciphertext pairs with that value and determine whether they satisfy the impossible differential characteristic.
4. Discard the values with which the pairs satisfy it from the key space. Go to step 3 unless the number of elements in the key space is almost one.

In the above algorithm the remaining element in the key space is the correct key value with high probability.

The block cipher Zodiac is submitted to the ISO/IEC JTC1/SC27–Korea in September, 2000. It consists of Feistel structure with 16 rounds. Its design principles are as follows [7]:

- **Simplicity:** We tried not to include ad-hoc design elements based on any obscure reason. We do not believe that more complicated design could give the more secure structure.
- **Provability:** As possible as we can do, we tried to design the cipher to have a provably secure structure especially resistant to the well known typical attacks such as differential cryptanalysis, linear cryptanalysis and interpolation attack.
- **Accommodation and Performance:** Our cipher was designed to get high performance both on general 32-bit CPU and general 8-bit microprocessor (e.g. smart cards).

The byte-wise diffusion in its round function F is very simple and cryptographically weak. Therefore, our observation is concentrated on impossible differential cryptanalysis of Zodiac [1]. In our work we show that there exist 15-round impossible differential characteristics in Zodiac. It follows that there exist r-round impossible differential characteristics if $r \leq 15$. We also give an efficient method to find the last round key using two 14-round impossible differential characteristics.

2 Zodiac

Zodiac follows the conventional Feistel structure except for the initial and final 64-bit data whitening by two 64-bit keys. The initial and final permutation $\prod$ is shown in Figure 2.

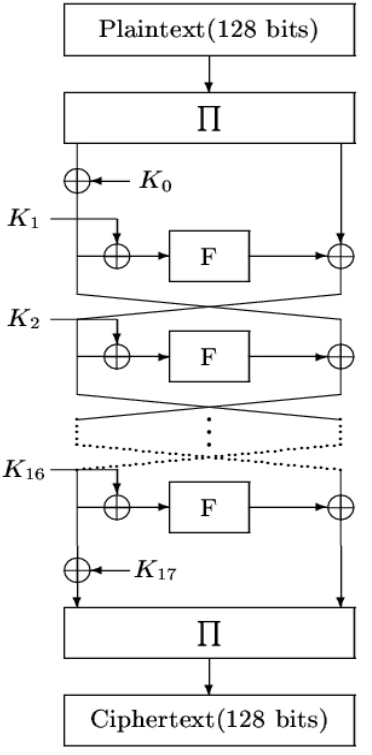


Fig. 1. The structure of Zodiac

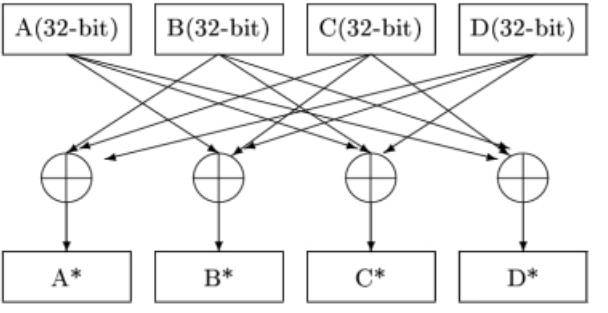


Fig. 2. Permutation $\prod$

[1] We also found two 15-round truncated differentials which can be useful for distinguishing the 15-round Zodiac from a random permutation, but impossible differential cryptanalysis succeeded in breaking the full Zodiac more efficiently.

Hereafter, we ignore the initial and final permutations and the whitening keys in the description of the attack because they don't affect on input or output difference values at all. Moreover, even in this case, the attack procedure and the required complexity is considered to be similar because whitening keys can be incorporated (XORed) into the internal round keys; i.e. K_0 can be XORed with the odd round keys, K_{17} can be XORed with the even round keys.

2.1 Round Function F

The round function F of Zodiac has very simple structure and all operations in F are XOR operation and 8-by-8 non-linear substitutions (S-boxes) i.e. table look-ups. The two S-boxes S1 and S2 are generated by the following functions $h(x)$ and $g(x)$, respectively:

$$h(x) = h_0(h_0(x)), \text{ where } h_0(x) = (45^x \bmod 257) \bmod 256$$
$$g(x) = (170 + x)^{-1} \text{ in GF}(2^8),$$

with irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. The detailed structure is shown in Figure 3.

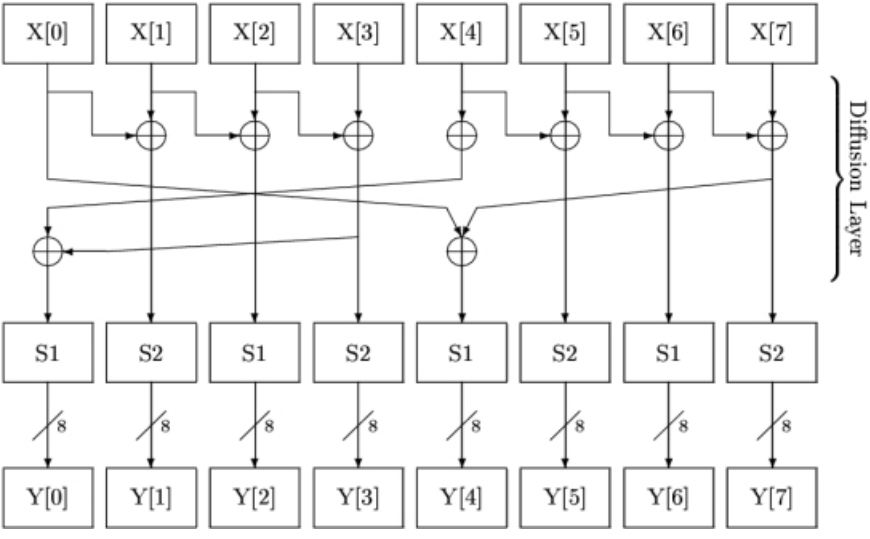


Fig. 3. Round function F

2.2 Key Schedule of Zodiac

Zodiac supports three types of key lengths, 128, 192, and 256 bits and needs sixteen 64-bit round keys and additionally two 64-bit keys for the input masking

and output masking of the processing data. The key scheduling algorithm of Zodiac is basically constructed from its round-function. For details, see [7]. We don't make use of the key schedule in our attack.

3 Notations

We use the following notations in this paper.

$I_L^i[j]$	the j-th byte in the left half of input difference of i-th round
$I_R^i[j]$	the j-th byte in the right half of input difference of i-th round
$O_L^i[j]$	the j-th byte in the left half of output difference of i-th round
$O_R^i[j]$	the j-th byte in the right half of output difference of i-th round
$S_I^i[j]$	the input difference of the j-th S-box (S1 or S2) of i-th round
$S_O^i[j]$	the output difference of the j-th S-box (S1 or S2) of i-th round
$K_i[j]$	the j-th byte in the round key of i-th round
$a, b, c, \ldots$	bytes with non-zero difference
$A, B, C, \ldots$	bytes with any (zero or non-zero) difference

4 Impossible Differential Characteristics of Zodiac

We found two 15-round impossible differential characteristics in Zodiac. Since the diffusion layer in the function F is performed by XORing two or three bytes, it is easy to compute zero-bytes in a difference and to find an impossible differential characteristic.

In Figure 4 if $(I_L^1, I_R^1) = (00000000, 00000aaa)$, then $I_L^8[3]$ must be zero, where $a, b, \ldots$ (small letters) denote nonzero one byte values and $A, B, \ldots$ (capital letters) denote any one byte values. Similarly, if $(O_L^{15}, O_R^{15}) = (00000a00, 00000000)$, then $O_R^8[3]$ must be nonzero. However, since $I_L^8 = O_R^8$, it is a contradiction. Therefore, this is a differential characteristic with zero probability, i.e. an impossible differential characteristic. In a similar way (and due to symmetry of the round function of Zodiac), we can see that the differential characteristic in Figure 5 is impossible.

Actually, these characteristics are impractical because there are too many zero bytes in the output differences of the 15th round. The more zero bytes are there, the fewer key-bits are derived and the more plaintext pairs are required for the attack. Nevertheless, the existence of 15-round impossible differential characteristics implies a cryptographically potential weakness of the block cipher Zodiac. It also follows that there exist r-round impossible differential characteristics if $r \leq 15$.

We also found two 15-round truncated differentials $(I_L^1, I_R^1) = (00000000, 00000a00)$, $(O_L^{15}, O_R^{15}) = (00000a00, 00000000)$ and $(I_L^1, I_R^1) = (00000000, 0a000000)$, $(O_L^{15}, O_R^{15}) = (0a000000, 00000000)$
with probability of about 2^{-120}. They may be useful for distinguishing the 15-round Zodiac from a random permutation, but failed to break the full Zodiac efficiently.

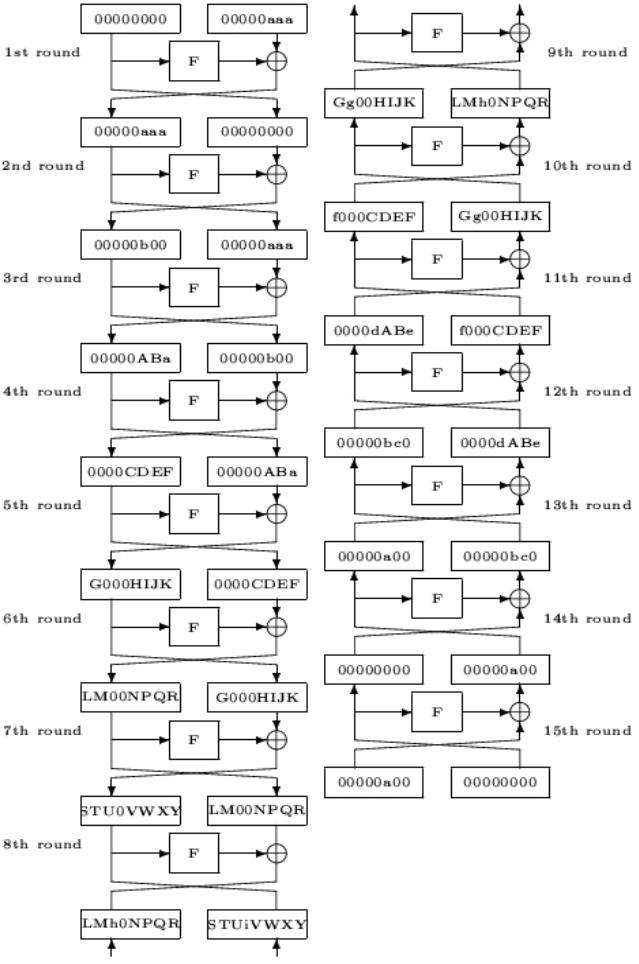


Fig. 4. 15-round impossible differential characteristic 15-I

5 Impossible Differential Cryptanalysis of Zodiac

Our attacks use two 14-round impossible characteristics shown in Figures 6 and 7 to derive the last round key. In our attack, we assume that the differential probability of the S-boxes is uniformly distributed and the round keys are random and uniformly distributed.

5.1 Attack on 15-Round Zodiac

First, we describe the attack on the 15-round Zodiac as a simple example. As we wrote in Section 2, we don't consider the whitening keys and let the round

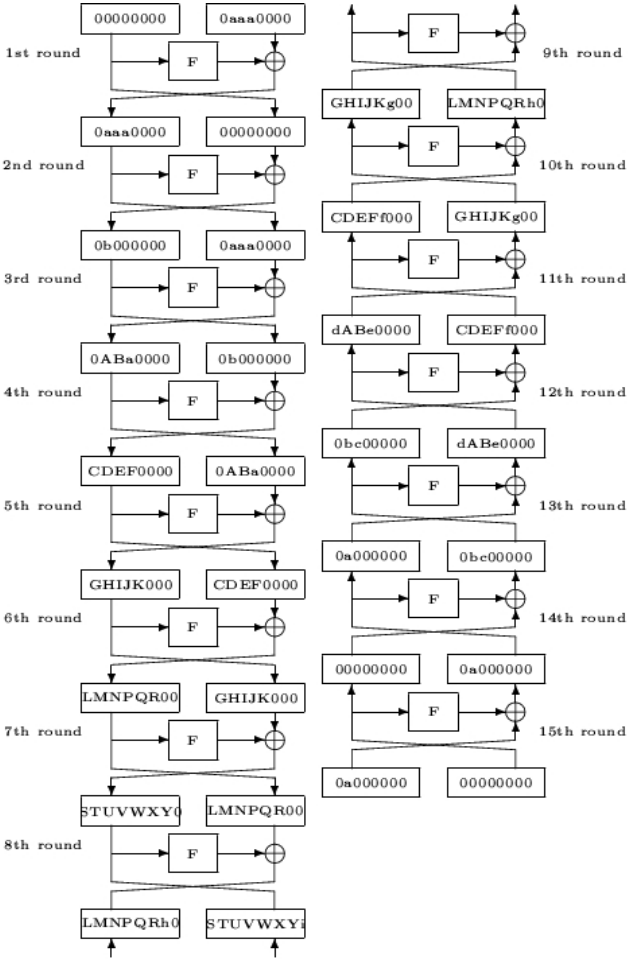


Fig. 5. 15-round impossible differential characteristic 15-II

keys $K_1, K_2, \ldots, K_{15}$. Our target is K_{15}. We define the key values K'_i to enter the S-boxes.

$$
\begin{aligned}
K'_i[0] &= K_i[2] \oplus K_i[3] \oplus K_i[4] \\
K'_i[1] &= K_i[0] \oplus K_i[1] \\
K'_i[2] &= K_i[1] \oplus K_i[2] \\
K'_i[3] &= K_i[2] \oplus K_i[3] \\
K'_i[4] &= K_i[0] \oplus K_i[6] \oplus K_i[7] \\
K'_i[5] &= K_i[4] \oplus K_i[5] \\
K'_i[6] &= K_i[5] \oplus K_i[6] \\
K'_i[7] &= K_i[6] \oplus K_i[7]
\end{aligned}
$$

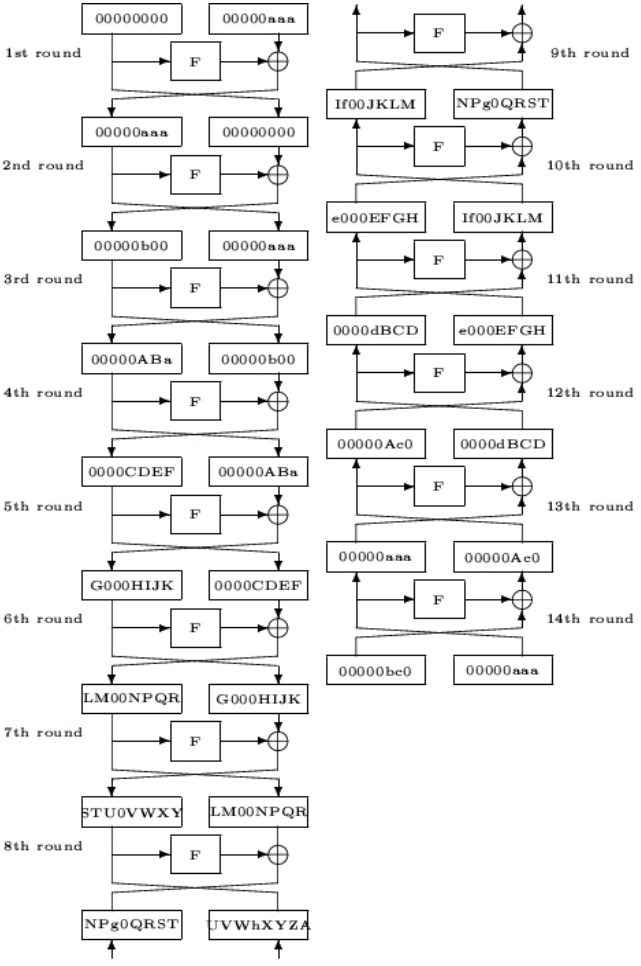


Fig. 6. 14-round impossible differential characteristic 14-I

We use the 14-round impossible differential shown in Figure 6. We choose a structure of 2^8 plaintext pairs with $(\Delta P_L, \Delta P_R) = (00000000, 00000aaa)$ and then, collect the pairs satisfying $\Delta C_L = 00000bc0, \Delta C_R[j] = 0 (0 \leq j \leq 3)$, and $\Delta C_R[4] \neq 0$ where the difference of the plaintexts $(\Delta P_L, \Delta P_R) = (I_L^1, I_R^1)$ and the difference of the ciphertexts $(\Delta C_L, \Delta C_R) = (O_R^{15}, O_L^{15})$. The probability that a plaintext pair satisfies these conditions is $(\frac{1}{2^8})^{10} \cdot (\frac{2^8-1}{2^8})^3 \simeq 2^{-80}$ (because there are 10 bytes with zero difference and 3 bytes with non-zero difference in $(\Delta C_L, \Delta C_R)$).

Since the input difference of F of the 15-th round is $00000bc0$, $S_I^{15}[4], S_I^{15}[5]$, and $S_I^{15}[7]$ are nonzero, and $S_I^{15}[6]$ takes any value. From the equation

$$S_O^{15}[4,5,6,7] \oplus \Delta C_R[4,5,6,7] = O_R^{14}[4,5,6,7],$$

an element in the key space of K_{15} is not a correct key value if $O_R^{14}[4] = 0$ and $O_R^{14}[5] = O_R^{14}[6] = O_R^{14}[7] \neq 0$ by computing with the ciphertexts and it. Thus, such elements should be eliminated from the key space. This method derives $4 \sim 7$th bytes of K'_{15}.

Let $\mathcal{K}$ be the key space of $K'_{15}[i](4 \leq i \leq 7)$. The probability that an element in $\mathcal{K}$ survives the test with such a pair is $1 - (\frac{1}{2^8} \cdot \frac{2^8-1}{2^8} \cdot \frac{1}{2^8} \cdot \frac{1}{2^8}) \simeq 1 - 2^{-24}$. Therefore, the number of the pairs required for narrowing down to one correct key is N such that $2^{32} \cdot (1 - 2^{-24})^N \simeq 1$. N is about $2^{28.5}$. Hence, the number of required chosen plaintext pairs is $2^{80} \cdot 2^{28.5} = 2^{108.5}$.

Since we can collect ${}_{2^8}C_2 = 2^{15}$ pairs from such one structure, we need $2^{93.5}$ structures. It follows that the attack requires $2^{93.5} \cdot 2^8 = 2^{101.5}$ chosen plaintexts.

This method is summarized as follows.
Goal: Finding $K'_{15}[4] \sim K'_{15}[7]$.

1. Collect sufficiently many ($2^{101.5}$) plaintext pairs from the structures whose ciphertext differences satisfy $\Delta C_L = 00000bc0, \Delta C_R[j] = 0(0 \leq j \leq 3)$, and $\Delta C_R[4] \neq 0$.
2. Choose a pair in the collection.
3. For each $k \in \mathcal{K}$,
   a) Compute $O_R^{14}[4], \ldots, O_R^{14}[7]$.
   b) Remove k from $\mathcal{K}$ if $O_R^{14}[4] = 0, O_R^{14}[5] = O_R^{14}[6] = O_R^{14}[7] \neq 0$.
   c) If $|\mathcal{K}| \leq \epsilon$, stop. Otherwise, go to 2. (ϵ is a sufficiently small integer.)

Using the characteristic in Figure 7, $K'_{15}[0] \sim K'_{15}[3]$ are also derived with $2^{101.5}$ chosen plaintext pairs. It is easy to compute K_{15} from K'_{15}.

5.2 Attack on 16-Round Zodiac

The impossible differential cryptanalysis for the 16-round Zodiac is a 2R-attack using two characteristics in Figures 6 and 7.

Similar to Section 5.1, we use the structures of 2^8 plaintexts satisfying $(\Delta P_L, \Delta P_R) = (00000000, 00000aaa)$ (with the characteristic in Figure 6) and collect the pairs whose ciphertext differences satisfy $\Delta C_L[j] = 0(0 \leq j \leq 3), \Delta C_L[4] \neq 0, \Delta C_R[0] \neq 0$, and $\Delta C_R[i] = 0(1 \leq i \leq 3)$ where the difference of the plaintexts $(\Delta P_L, \Delta P_R) = (I_L^1, I_R^1)$ and the difference of the ciphertexts $(\Delta C_L, \Delta C_R) = (O_R^{16}, O_L^{16})$. Therefore, the probability that any plaintext pair satisfies these conditions is $(\frac{1}{2^8})^7 \cdot (\frac{2^8-1}{2^8})^2 \simeq 2^{-56}$. Let $\mathcal{K}_1$ be the key space of $K'_{16}[0]$ and $K'_{16}[i](4 \leq i \leq 7)$ and let $\mathcal{K}_2$ be the key space of $K'_{15}[j](4 \leq j \leq 7)$. This attack is summarized as follows.
Goal: Finding $K'_{16}[0], K'_{16}[i](4 \leq i \leq 7)$, and $K'_{15}[j](4 \leq j \leq 7)$

1. Collect sufficiently many (discussed later) plaintext pairs from the structures whose ciphertext differences satisfy $\Delta C_R[0] \neq 0, \Delta C_R[i] = 0(1 \leq i \leq 3), \Delta C_L[j] = 0(0 \leq j \leq 3)$, and $\Delta C_L[4] \neq 0$.
2. Choose a pair in the collection.
3. For each $k_1 \in \mathcal{K}_1$,

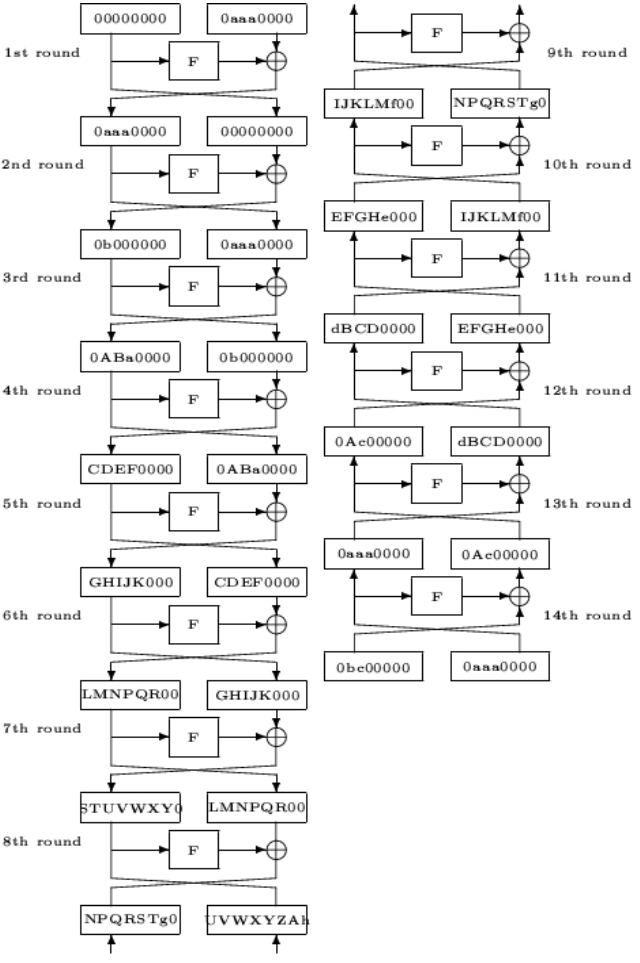


Fig. 7. 14-round impossible differential characteristic 14-II

a) Compute $O_R^{15}[0], O_R^{15}[l] (4 \leq l \leq 7)$.

b) Unless $O_R^{15}[0] = O_R^{15}[4] = O_R^{15}[7] = 0$, $O_R^{15}[5] \neq 0$, and $O_R^{15}[6] \neq 0$, choose another $k_1 \in \mathcal{K}_1$ and go to 3.(a).

c) For each $k_2 \in \mathcal{K}_2$,

   i. Compute $O_R^{14}[s] (4 \leq s \leq 7)$.

   ii. Remove k_1 and k_2 from $\mathcal{K}_1$ and $\mathcal{K}_2$, respectively, if $O_R^{14}[4] = 0$ and $O_R^{14}[5] = O_R^{14}[6] = O_R^{14}[7] \neq 0$.

4. If $|\mathcal{K}_1| \leq \epsilon$ and $|\mathcal{K}_2| \leq \epsilon'$, stop. Otherwise, go to 2. (ϵ and ϵ' are small integer.)

We obtain 72 bits of K'_{15} and K'_{16}. The probability that any $(k_1, k_2) \in \mathcal{K}_1 \times \mathcal{K}_2$ survives the test with such a pair is

$$1-\left\{\left(\frac{1}{2^8}\right)^2 \cdot \left(\frac{2^8-1}{2^8}\right)^2 \cdot \frac{1}{2^8-1}\right\} \cdot \left\{\left(\frac{1}{2^8}\right)^3 \cdot \frac{2^8-1}{2^8}\right\} \simeq 1-\left(\frac{1}{2^8}\right)^6 = 1-2^{-48}.$$

Thus, the number of the pairs collected for the attack is N such that $2^{72} \cdot (1 - 2^{-48})^N \simeq 1$. N is about $2^{53.6}$. Therefore, the required number of chosen plaintext pairs is $2^{56} \cdot 2^{53.6} = 2^{109.6}$.

Since we can collect 2^{15} pairs from one structure, we need $2^{94.6}$ structures. It follows that the attack requires $2^{94.6} \cdot 2^8 = 2^{102.6}$ chosen plaintexts.

Using the characteristic in Figure 7, we obtain $K'_{16}[i](0 \leq i \leq 4)$ and $K'_{15}[j](0 \leq j \leq 3)$ with the same complexity, and then do K_{15} and K_{16} from K'_{15} and K'_{16}. The time complexity of this attack is computed as follows.

- $2^{103.6} (= 2 \times 2^{102.6})$ encryptions of chosen plaintexts
- Finding K'_{15} and K'_{16} : $2 \cdot \{2^{72} + 2^{72} \cdot (1-2^{-48}) + \cdots + 2^{72} \cdot (1-2^{-48})^{2^{53.6}-1}\} \simeq 2^{121} - 2^{49}$ two-round encryptions. $\Rightarrow 2^{118} - 2^{46}$ encryptions.

We can find every round key similarly to the above method. The total time complexity is about 2^{119} at most, since complexity decreases as the number of rounds reduces.

6 Conclusion

The design principles of Zodiac are the simplicity, provability, accommodation and performance. By the simplicity Zodiac attains the accommodation and good performance. However, Zodiac has crucial weakness in security from the theoretical point of view because of its poor design of the diffusion layer.

In this paper we did find the 15-round and 14-round impossible differentials of Zodiac. Using two 14-round impossible differential we attacked the 15-round Zodiac with $2^{102.5}$ chosen plaintext pairs and the full-round Zodiac with $2^{103.6}$ chosen plaintext pairs. The complexity of our full round attack is at most 2^{119} encryption, which means that this attack is more effective than the exhaustive key search attack. Furthermore, when the key length is 192-bit or 256-bit, its security decreases more seriously.

We think the diffusion layer should be changed. This result shows the importance of the design for the diffusion layer as well as the substitution layer.

References

1. I. Ben-Aroya and E. Biham, *Differential Cryptanalysis of Lucifer*, Journal of Cryptology, vol.9, no.1, pp.21–34, 1996.
2. E. Biham, A. Biryukov, and A. Shamir, *Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials*, Advances in Cryptology — EUROCRYPT'99, LNCS 1592, Springer-Verlag, 1999, pp.12–23.

3. E. Biham and A. Shamir, *Differential cryptanalysis of DES-like cryptosystems*, Advances in Cryptology — CRYPTO'90, LNCS 537, Springer-Verlag, 1991, pp.2–21.
4. L. Brown, J. Pieprzyk, and J. Seberry, *LOKI - A cryptographic primitive for authentication and secrecy applications*, Advances in Cryptology — AUSCRYPT'90, LNCS 453, pp.229–236, Springer-Verlag, 1990.
5. L. R. Knudsen, *Truncated and Higher Order Differential*, Fast Software Encryption Workshop 94, LNCS 1008, pp.229–236, Springer-Verlag, 1995.
6. L. R. Knudsen and T. Jakobsen, *The Interpolation Attack on Block Ciphers*, Fast Software Encryption Workshop 97, LNCS 1267, pp. 28–40, Springer-Verlag, 1997.
7. ChangHyi Lee, KyungHwa Jun, MinSuk Jung, SangBae Park, and JongDeok Kim, *Zodiac Version 1.0(revised) Architecture and Specification*, Standardization Workshop on Information Security Technology 2000, Korean Contribution on MP18033, ISO/IEC JTC1/SC27 N2563, 2000, Available at the KISA's web page, http://www.kisa.or.kr/seed/index.html.
8. S. Moriai, T. Shimoyama, and T. Kaneko, *Higher Order Differential Attack of a CAST cipher*, Fast Software Encryption Workshop 98, LNCS 1372, pp.17–31, Springer-Verlag, 1998.
9. B. Van Rompay, L. R. Knudsen, and V. Rijmen, *Differential cryptanalysis of the ICE encryption algorithm*, Fast Software Encryption Workshop 98, LNCS 1372, pp.270–283, Springer-Verlag, 1998.
10. A. Shimizu and S. Miyaguchi, *Fast Data Encipherment Algorithm FEAL*, Advances in Cryptology — EUROCRYPT'87, LNCS 304, pp.267–278, Springer-Verlag, 1988.
11. D. Wagner, *The boomerang attack*, Fast Software Encryption Workshop 99, LNCS 1636, pp.156–170, Springer-Verlag, 1999.

The Block Cipher SC2000

Takeshi Shimoyama[1], Hitoshi Yanami[1], Kazuhiro Yokoyama[1], Masahiko Takenaka[2], Kouichi Itoh[2], Jun Yajima[2], Naoya Torii[2], and Hidema Tanaka[3]

[1] Fujitsu Laboratories LTD.
4-1-1, Kamikodanaka Nakahara-ku Kawasaki 211-8588, Japan
{shimo,yanami,kayoko}@flab.fujitsu.co.jp
[2] Fujitsu Laboratories LTD.
64, Nishiwaki, Ohkubo-cho, Akashi 674-8555, Japan
{takenaka,kito,jyajima,torii}@flab.fujitsu.co.jp
[3] Science University of Tokyo
Yamazaki, Noda, Chiba, 278 Japan
tanaka@ee.noda.sut.ac.jp

Abstract. In this paper, we propose a new symmetric key block cipher SC2000 with 128-bit block length and 128-,192-,256-bit key lengths. The block cipher is constructed by piling two layers: one is a Feistel structure layer and the other is an SPN structure layer. Each operation used in two layers is S-box or logical operation, which has been well studied about security. It is a strong feature of the cipher that the fast software implementations are available by using the techniques of putting together S-boxes in various ways and of the Bitslice implementation.

1 Introduction

In this paper, we propose a new block cipher SC2000. The algorithm has 128-bit data inputs and outputs and 128-bit, 192-bit, and 256-bit keys can be used. Our design policy for the cipher is to enable high-speed in software and hardware processing on various platforms while maintaining the high-level security as cipher. The cipher is constructed by piling a Feistel and an SPN structures with S-boxes for realizing those properties.

For evaluation of security, we inspect the security in the design against differential attacks, linear attacks, higher order differential attacks, interpolation attacks, and so on. We employ a structure that facilitates security verification by using modules with established reputation concerning security. In S-box design, we use composite maps of an exponentiation over a Galois field and affine transformations[8]. For the diffusion layer in the inner function of Feistel structure, we use an MDS matrix[4,5,7,10]. We set the number of rounds as for having a sufficiently good security margin.

As high-speed implementation technology, we designed nonlinear arithmetic operations to enable high-speed implementation on a wide range of processors. Individual implementations are realized by selecting optimum parameters corresponding to the register lengths and cache memory sizes. Moreover, for the

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 312–327, 2002.

fast software encryption of SPN structure, we adopted a structure that permits the use of latest high-speed implementation called Bitslice[1,2,3,9]. In hardware implementation, we aimed at a structure that enables extremely compact implementation by using nonlinear arithmetic and logical units with up to 6-bit input/output in the encryption.

2 Preliminary

Basically, the following rules are used for notation:

- *Big-Endian* is used as *Endian*, where the highest bit in each value a is the 0-th bit of a. For example, $a = 10$ (decimal (4-bit) integer) is expressed as $(a_0, a_1, a_2, a_3) = (1, 0, 1, 0)$.
- The number of bits in a variable is expressed by each variable superscripted with a decimal number enclosed by parentheses as $a^{(4)}$. For 32-bit variables, the number of bits is omitted as a.
- A number without a prefix expresses a decimal number, and a number prefixed with `0x` expresses a hexadecimal number like as `0x01234567`.
- XOR is the exclusive-or of two variables and expressed as $a \oplus b$.
- AND is the logical product of two variables and expressed as $a \wedge b$.
- OR is the logical sum of two variables and expressed as $a \vee b$.
- NOT is an operation that inverts all bits and expressed as $\bar{b}$.
- ADD is an operation that applies modulo 2^{32} operation $(a + b \pmod{2^{32}})$ to the result of addition of two 32-bit variables and expressed as $a \boxplus b$.
- SUB is an operation that applies modulo 2^{32} operation $(a - b \pmod{2^{32}})$ to the result of subtraction of two 32-bit variables and expressed as $a \boxminus b$.
- MUL is an operation that applies modulo 2^{32} operation $(a \times b \pmod{2^{32}})$ to the result of multiplication of two 32-bit variables and expressed as $a \boxtimes b$.
- Rotate left one bit is an operation that rotates a 32-bit variable left by 1 bit $((a_0, a_1, \cdots, a_{31}) \rightarrow (a_1, a_2, \cdots, a_0))$ and expressed as $a \lll_1$.

3 Algorithm Specifications

3.1 Encryption Function

The encryption function consists of the following three functions: I function, B function, and R function, each of which has a (32 bits $\times$ 4) input/output. Among these three functions, the I function XORs the key and the B and R functions stir the data. The encryption function for a 128-bit key consists of 14 rounds of the I function, which XORs the key, and as data randomizing, 7 rounds of the B function and 12 rounds of the R function, totaling 19 rounds. The encryption function for a 192-bit or 256-bit key consists of 16 rounds of the I function, which XORs the key, and as data randomizing, 8 rounds of the B function and 14 rounds of the R function, totaling 22 rounds. Each function is executed in the order of I-B-I-$R \times R \cdots$ repeatedly. (See the table below.) For a

128-bit key, fifty-six 32-bit extended keys are used; for a 192-bit or 256-bit key, sixty-four 32-bit extended keys are used.

The following lists the entire configuration of the encryption function. Symbols used in the configuration are as follows: (See also Fig.1.)

I	I function
B	B function
$R5$	R function with $mask =$ `0x55555555`
$R3$	R function with $mask =$ `0x33333333`
$-$	Straight connection $(a, b, c, d) \rightarrow (a, b, c, d)$
$\times$	Cross connection $(a, b, c, d) \rightarrow (c, d, a, b)$

Configuration for 128-bit key:
(in)-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-(out)
Configuration for 192-bit or 256-bit key:
(in)-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-$R3 \times R3$-I-B-I-$R5 \times R5$-I-B-I-(out)

3.2 Decryption Function

The decryption function consists of the following three functions: I function, B^{-1} function, and R function, each of which has a (32 bits $\times$ 4) input/output. Among these three functions, the I function and R function is as same as one in the Encryption function, and the B^{-1} functin is the inverse function of the B function. Each function is executed in the order of I-B^{-1}-I-$R \times R \cdots$ repeatedly.

The following lists the entire configuration of the decryption function. Symbols used in the configuration are as follows: (See Fig.1.)

I	I function
B^{-1}	B^{-1} function
$R5$	R function with $mask =$ `0x55555555`
$R3$	R function with $mask =$ `0x33333333`
$-$	Straight connection $(a, b, c, d) \rightarrow (a, b, c, d)$
$\times$	Cross connection $(a, b, c, d) \rightarrow (c, d, a, b)$

Configuration for 128-bit key:
(in)-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-(out)
Configuration for 192-bit or 256-bit key:
(in)-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-$R3 \times R3$-I-B^{-1}-I-$R5 \times R5$-I-B^{-1}-I-(out)

3.3 *I* Function

The I function is given four 32-bit variables and four 32-bit extended keys as inputs, and it outputs four 32-bit variables. Each input data is XORed with the extended key. The extended keys are inputs only to the I functions.

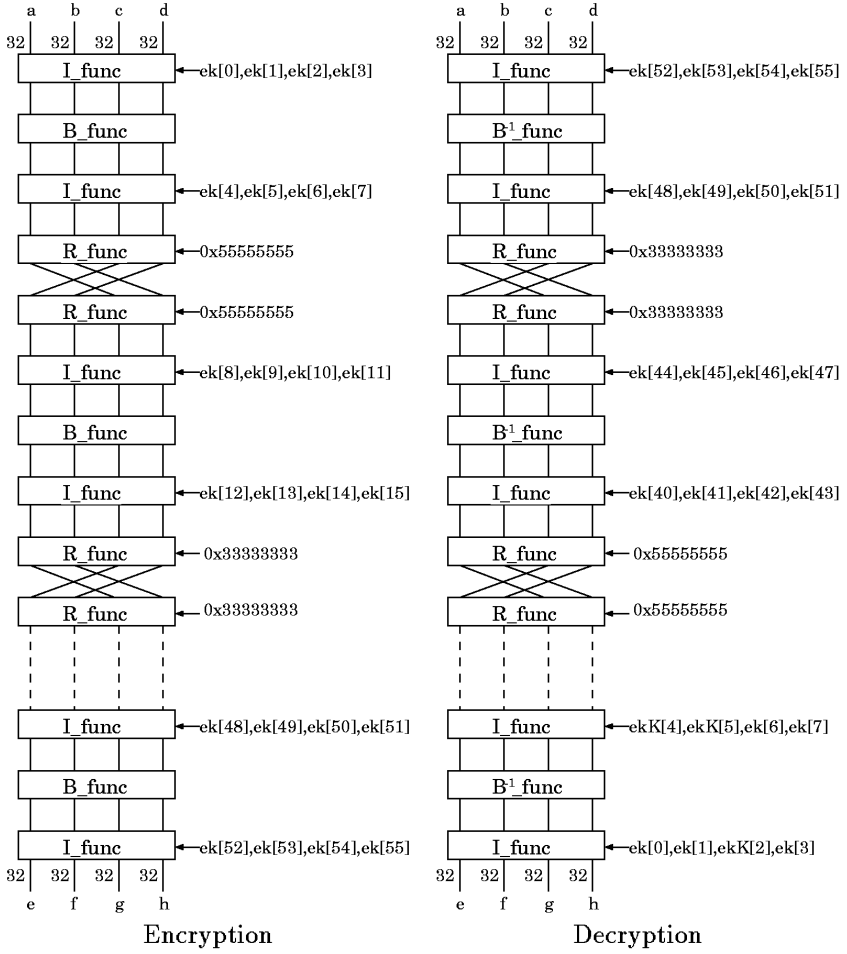


Fig. 1. Encryption and Decryption function (128-bit key)

3.4 *R* Function

The R function is a Feistel-type data randomizing function having four 32-bit variables as the input and the output. The R function inputs the third entry and the fourth entry (c, d) of the input data and the constant data mask to the F function, and it XORs the two outputs from the F function with the first entry and the second entry (a, b) of the input data. (See Fig.2.)

***F* function.** The F function is given two 32-bit variables and constant as inputs, and outputs two 32-bit variables. The F function processes two variables with the S function and the M function respectively and then processes the two outputs with the L function.

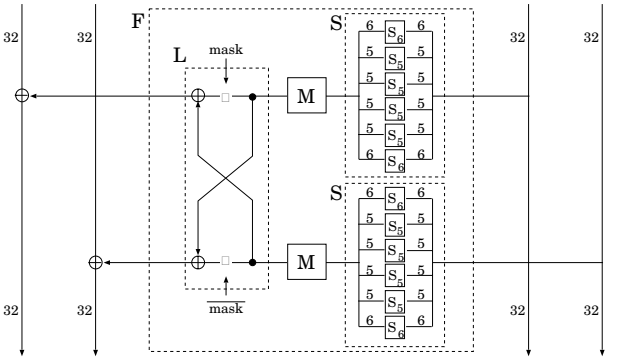


Fig. 2. R function

***S* function.** The S function splits the 32-bit input into 6 bits, 5 bits, 5 bits, 5 bits, 5 bits and 6 bits. The S function then looks up the 6-bit S-box table S_6 with the original 6 bits; it looks up the 5-bit S-box table S_5 with the 5 bits. The function then aligns individual outputs in the same order to generate 32 bits.

Each S-box is a composition of an exponentiation over a Galois field and an affine map, whose maximum differential/linear probability is 2^{-4}, and the total degree is 3 for S_5, and is 5 for S_6. See Section 4 for the elements of the tables S_5 and S_6.

***M* function.** The M function is a 32-bit input/output bitwise linear function. The M function considers the input a as a vector of 32 entries of 1 bit and outputs $a_0 \cdot \mathtt{M}[0] \oplus \cdots \oplus a_{31} \cdot \mathtt{M}[31]$. See Section 4 for the elements of the matrix `M`.

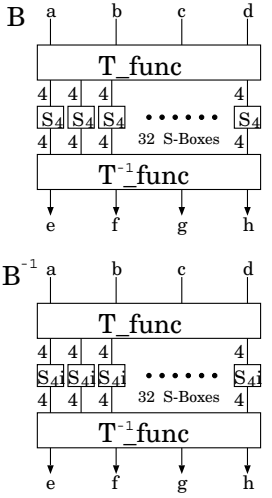
This function is designed as an MDS matrix with respect to the S function. The summation of the number of an input active S-box and the number of the output active S-box correspond to the input is greater than or equal to 6.

***L* function.** The L function is given two 32-bit variables as inputs and outputs two 32-bit variables, and a constant. The L function outputs two data $((a \wedge mask) \oplus b, (b \wedge (\overline{mask})) \oplus a)$ by processing two input values (a, b). As the $mask$ values used in the L function, there are two values: `0x55555555` and `0x33333333` for increasing the security.

Software implemenation techniques of the *F* function. In the description of the F function, the S function, the M function and the L function are executed individually in this order. However, the S and the M functions can be composed because the M function is a linear map. In the explanation of the S function, 5-bit and 6-bit S-box tables are looked up by (6,5,5,5,5,6). However, executing the table lookups by (6,10,10,6) or (11,10,11), which is obtained by jointing two consecutive S-boxes into a new one, speeds up the operations because the number

of table lookups is reduced. In this case, creating the table $M \circ S_6$(6-bit input, 32-bit output) and $M \circ (S_5 S_5)$(10-bit input, 32-bit output) by composing two tables can reduce the number of table lookups, leading to a speedup of operations. Also, creating the table $M \circ (S_6 S_5)$ and $M \circ (S_5 S_6)$ (11-bit input, 32-bit output) or composing the L function can realize a further speedup in the processor with a larger cashe memory.

Implemenataion	# of Table Lookups	Size of Tables
(6,5,5,5,5,6)	6	1 KByte
(6,10,10,6)	4	9 KByte
(11,10,11)	3	20 KByte



$B\_func()$

$t1 = \overline{b}$
$t2 = a \vee d$
$t3 = c \oplus t2$
$t4 = t1 \wedge t3$
$e = d \oplus t4$
$t6 = t2 \oplus e$
$t7 = a \oplus b$
$t8 = t6 \vee t7$
$f = c \oplus t8$
$t10 = c \wedge t3$
$t11 = t1 \oplus t10$
$t12 = c \oplus t4$
$t13 = t6 \wedge t12$
$h = t7 \oplus t13$
$t15 = c \vee h$
$g = t11 \oplus t15$

$B^{-1}\_func()$

$t1 = c \oplus d$
$t2 = t1 \vee b$
$t3 = t2 \oplus c$
$t4 = t3 \wedge a$
$t5 = \overline{c}$
$t6 = t5 \vee b$
$t7 = t6 \oplus d$
$e = t7 \oplus t4$
$t8 = a \oplus t6$
$t9 = t8 \vee d$
$g = t9 \oplus b$
$t10 = c \oplus g$
$t11 = t10 \vee t4$
$t12 = t11 \wedge t9$
$f = t12 \oplus e$
$t13 = t6 \oplus g$
$t14 = t13 \wedge t1$
$h = t14 \oplus a$

Fig. 3. B/B^{-1} function and an example of Bitslice implementation

3.5 B/B^{-1} Function

The B function converts four 32-bit inputs into thirty two 4-bit values with the T function, each of which is processed with the same 4-bit S-box table S_4. This function then recover the output values into four 32-bit values and outputs those values with the T^{-1} funciton. (See Fig.3.) The B^{-1} function uses $S_4 i$ which is the inverse of the 4-bit S-box table S_4. See Section 4 for the elements of the tables S_4 and $S_4 i$.

The B/B^{-1} function can execute Bitslicing by expressing the S_4 table in logical form[1,2]. Bitslicing enables an aggregate processing of thirty-two S-box lookups, leading to calculation speedup. For example, the B and B^{-1} function can be expressed in logical form as shown in the Fig. 3.

T/T^{-1} function. The T function converts four 32-bit inputs to the (4,32)-matrix, and it converts the matrix into the transposed (32,4)-matrix, and then converts it into thirty two 4-bit data values. The T^{-1} function is the inverse function of T.

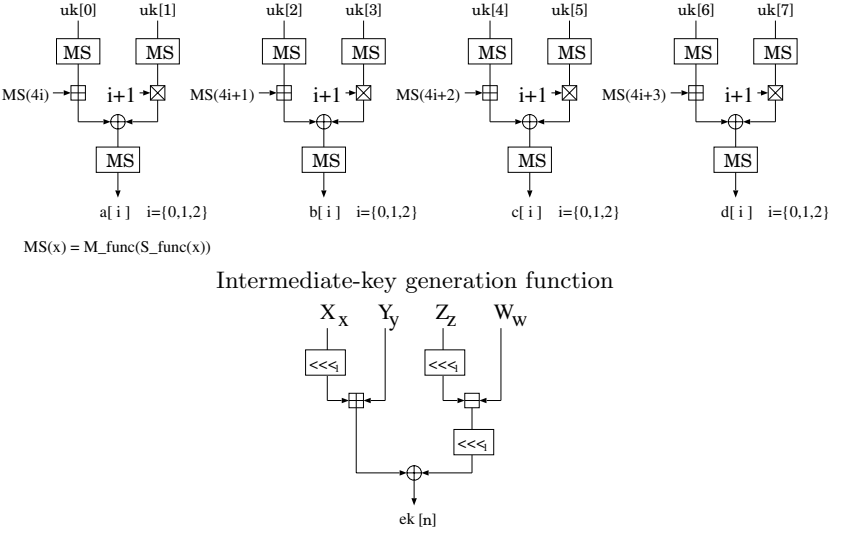


Fig. 4. Key Schedule

3.6 Key Schedule Specifications

The key schedule generates fifty-six 32-bit extended keys (for 128-bit key) or sixty-four 32-bit extended keys (for 192-bit or 256-bit key) from the secret key. The key schedule is a function consisting of the intermediate-key generation function and the extended-key generation function. For a 256-bit key, the key schedule splits the 256-bit secret key into eight 32-bit user keys $uk[0], \ldots, uk[7]$. For a 192-bit key, the key schedule splits the 192-bit secret key into six 32-bit user keys $uk[0], \ldots, uk[5]$ like as 256-bit key, then the function expands the 6 user keys to 8 user keys by $uk[6] = uk[0]$ and $uk[7] = uk[1]$. In the same way, for a 128-bit key, the key schedule splits the 128-bit secret key into four 32-bit user keys $uk[0], \ldots, uk[3]$, then expands the 4 user keys to 8 user keys by $uk[4] = uk[0]$, $uk[5] = uk[1]$, $uk[6] = uk[2]$ and $uk[7] = uk[3]$. From the user keys $uk[]$, it generates the intermediate keys $imkey[]$ with the intermediate-key generation function and then executes the extended-key generation function to generate 32-bit extended keys $ek[]$.

Intermediate-key Generation Function. This function generates twelve 32-bit intermediate keys $a[i], b[i], c[i], d[i]$ for $i \in \{0, 1, 2\}$ from eight 32-bit user keys by the following processing.

for $(i = 0; i < 3; i++)$ {
$a[i] = M(S((M(S(4i)) \boxtriangle M(S(uk[0])))\oplus (M(S(uk[1])) \boxminus (i+1))))$
$b[i] = M(S((M(S(4i+1)) \boxtriangle M(S(uk[2])))\oplus (M(S(uk[3])) \boxminus (i+1))))$
$c[i] = M(S((M(S(4i+2)) \boxtriangle M(S(uk[4])))\oplus (M(S(uk[5])) \boxminus (i+1))))$
$d[i] = M(S((M(S(4i+3)) \boxtriangle M(S(uk[6])))\oplus (M(S(uk[7])) \boxminus (i+1))))$
}

Extended-key Generation Function. This function generates fifty-six 32-bit extended keys from twelve 32-bit intermediate keys for a 128-bit key; and also generates sixty-four 32-bit extended keys for a 192-bit or 256-bit key.

if $(KeyLength == 128)$ $num\_ekey = 56$ else $num\_ekey = 64$
for $(n = 0; n < num\_ekey; n++)$ {
$s = n \pmod 9$
$t = (n + \lfloor n/36 \rfloor) \pmod{12}$
$X = Order[t][0]$ $Y = Order[t][1]$ $Z = Order[t][2]$ $W = Order[t][3]$
$x = Index[s][0]$ $y = Index[s][1]$ $z = Index[s][2]$ $w = Index[s][3]$
$ek[n] = ((X[x] \lll n_1) \boxtriangle Y[y]) \oplus (((Z[z] \lll n_1) \ggg W[w]) \lll n_1)$
}

4 Table

The following shows the tables used by functions for the cipher:

```
S6[64]  = {47,59,25,42,15,23,28,39,26,38,36,19,60,24,29,56,
           37,63,20,61,55, 2,30,44, 9,10, 6,22,53,48,51,11,
           62,52,35,18,14,46, 0,54,17,40,27, 4,31, 8, 5,12,
            3,16,41,34,33, 7,45,49,50,58, 1,21,43,57,32,13};
S5[32]  = {20,26,7,31,19,12,10,15,22,30,13,14, 4,24, 9,18,
           27,11, 1,21, 6,16, 2,28,23, 5, 8, 3, 0,17,29,25};
S4[16]  = { 2, 5,10,12, 7,15,1,11,13, 6, 0, 9, 4, 8, 3,14};
S4i[16] = {10, 6, 0,14,12, 1, 9,4,13,11, 2, 7, 3, 8,15, 5};
M[32] = {0xd0c19225,0xa5a2240a,0x1b84d250,0xb728a4a1,
         0x6a704902,0x85dddbe6,0x766ff4a4,0xecdfe128,
         0xafd13e94,0xdf837d09,0xbb27fa52,0x695059ac,
         0x52a1bb58,0xcc322f1d,0x1844565b,0xb4a8acf6,
         0x34235438,0x6847a851,0xe48c0cbb,0xcd181136,
         0x9a112a0c,0x43ec6d0e,0x87d8d27d,0x487dc995,
         0x90fb9b4b,0xa1f63697,0xfc513ed9,0x78a37d93,
         0x8d16c5df,0x9e0c8bbe,0x3c381f7c,0xe9fb0779};
```

Order

t	0	1	2	3	4	5	6	7	8	9	10	11
X	a	b	c	d	a	b	c	d	a	b	c	d
Y	b	a	d	c	c	d	a	b	d	c	b	a
Z	c	d	a	b	d	c	b	a	b	a	d	c
W	d	c	b	a	b	a	d	c	c	d	a	b

Index

s	0	1	2	3	4	5	6	7	8
x	0	1	2	0	1	2	0	1	2
y	0	1	2	1	2	0	2	0	1
z	0	1	2	0	1	2	0	1	2
w	0	1	2	1	2	0	2	0	1

Table 1. Distribution tables for differential/linear approximations of S_4

differential

ΔIn \ ΔOut	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1	0	0	0	0	0	0	2	2	2	2	2	2	2	2	0	0
0x2	0	0	0	0	2	0	4	2	2	2	0	0	0	2	0	2
0x3	0	0	0	0	2	0	2	0	0	0	2	2	2	0	4	2
0x4	0	0	0	2	0	2	0	4	0	2	2	2	0	0	2	0
0x5	0	2	4	0	0	2	0	0	0	0	2	0	0	4	2	0
0x6	0	2	0	4	2	0	0	0	2	0	0	0	0	2	4	0
0x7	0	0	0	2	2	4	0	0	2	2	0	2	0	2	0	0
0x8	0	0	2	4	0	4	0	2	0	0	2	0	0	0	0	2
0x9	0	0	0	2	2	0	0	0	4	0	0	2	2	0	0	4
0xa	0	2	2	2	2	2	0	2	0	0	2	0	2	0	0	0
0xb	0	2	0	0	0	2	0	0	0	4	0	2	4	0	0	2
0xc	0	2	4	0	0	0	2	0	0	4	2	0	0	2	0	0
0xd	0	4	2	0	2	0	0	0	2	0	2	2	0	0	0	2
0xe	0	2	0	0	2	0	2	2	0	0	0	2	2	2	2	0
0xf	0	0	2	0	0	0	4	2	2	0	0	0	2	0	2	2

$(\#\{In|\Delta In \to \Delta Out\})$

linear

ΓIn \ ΓOut	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1	0	0	0	0	2	2	-2	-2	4	0	0	4	2	-2	2	-2
0x2	0	0	0	0	-4	4	0	0	2	2	-2	-2	-2	-2	-2	-2
0x3	0	0	0	0	-2	-2	-2	-2	-2	2	-2	2	0	4	0	-4
0x4	0	2	2	-4	0	2	-2	0	0	2	-2	0	0	2	2	4
0x5	0	2	-2	0	2	4	0	2	-4	2	2	0	2	0	0	-2
0x6	0	-2	-2	-4	0	-2	-2	4	2	0	0	-2	2	0	0	-2
0x7	0	-2	2	0	2	0	0	-2	-2	0	-4	-2	4	-2	-2	0
0x8	0	-2	-2	0	0	2	-2	-4	0	-2	2	-4	0	2	2	0
0x9	0	2	-2	-4	2	0	4	-2	0	-2	-2	0	-2	0	0	-2
0xa	0	-2	-2	0	0	2	2	0	2	0	0	2	2	4	-4	2
0xb	0	2	-2	4	2	0	0	2	2	0	-4	-2	0	2	2	0
0xc	0	4	0	0	0	0	-4	0	0	-4	0	0	0	0	-4	0
0xd	0	0	-4	0	2	-2	-2	-2	0	4	0	0	-2	-2	-2	2
0xe	0	0	4	0	4	0	0	0	2	2	2	-2	-2	2	-2	-2
0xf	0	4	0	0	-2	-2	2	-2	2	2	2	-2	4	0	0	0

$(\#\{In|In{\cdot}\Gamma In \oplus Out{\cdot}\Gamma Out = 0\} - 8)$

5 Security Evaluation Policy

Since differential attacks and linear attacks were presented, designers of block cipher need to consider the security specifically against these attacks among standard attacks, and trials to show the security against these attacks by several methods have been made.

While a method of finding the theoretical upper bounds of security can show the security of cipher theoretically, such a method has certain constraints in the structure of the cipher to apply the theory. The block cipher SC2000 cannot be directly applied those techniques because the cipher uses a *combined* structure of the Feistel structure and the SPN structure. It is also difficult to construct a new technique to evaluate the security of the structure of the cipher. Therefore, we employ the following method to show the security of the cipher:

> Search for characteristics that give differential probability or linear probability value greater than or equal to a certain threshold value, and then show that such characteristics do not exist.

6 The Security against Differential Attack

In this session, we describe an evaluation of the security against differential attack. Here we study the cases where the input differential of the F function in the R function is repeated by two rounds by the effect of the B function. Since the structure of SC2000 is constructed by the iteration of 3-round non-linear functions B-R-R, it is expected that the *effective* differential characteristics are obtained from the concatenation of 3-round characteristics.

In the B function, each S-box S_4 may sometimes give effects other than exchanging upper and lower 2-bit differentials. For example, in the differential

Table 2. Differential characteristic probabilities for each round

round	1	2	3	4	5	6	7	8	9	10	11	12	13
	B	R_5	R_5	B	R_3	R_3	B	R_5	R_5	B	R_3	R_3	B
active S-box	5	0	4	5	0	4	5	0	4	5	0	4	5
probability($\log_2$)	-15	-15	-33	-48	-48	-66	-81	-81	-99	-114	-114	-132	-147

distribution table for S_4, (0x6,0x8)- and (0x2,0x8)-entry have nonzero probabilities. (See Table 1.) Therefore, the B function in which thirty-two S-boxes S_4 are aligned may possibly have the following input/output differential:

$$(\Delta a, 0, 0, 0) \xleftarrow{B} (\Delta c, \Delta c, \Delta a, 0),$$

$\Delta d \prec \Delta a, \Delta c = mask \wedge \Delta d$, where $mask$ is the constant value used in the R function, and $\Delta d \prec \Delta a$ means $\Delta d \vee \Delta a = \Delta a$. By using this, the following differential may pass through: (see Figure 5.)

$$\begin{aligned}(\Delta a, 0, 0, 0) &\xleftarrow{B} (\Delta c, \Delta d, \Delta a, 0)\\ (0, 0) &\xleftarrow{F} (0, 0)\\ (\Delta c, \Delta d) &\xleftarrow{F} (\Delta a, 0)\\ (\Delta a, 0, 0, 0) &\xleftarrow{B} (\Delta c, \Delta d, \Delta a, 0).\end{aligned}$$

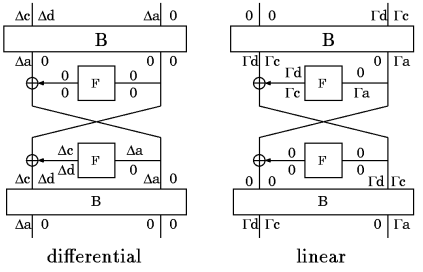


Fig. 5. 3-round cycle differential/linear characteristics

Since the (0xe,0x8)-entry of the differential distribution table is zero, Δc must be 0. Also in the example above, we set lowermost 32 bits in the input differential of the F function zero, but it is also necessary to search for cases where both upper and lower bits of the input differential of the F function are nonzero. This types of characteristics may be formed depending on the entry of the differential distribution table of S_4.

As a result of calculations of differentials in this form by computer, we obtained the following pattern which is one of the best differential characteristic whose probability is 2^{-33} for 3-round B-R-R described above. (See Table 2.)

Table 3. Linear characteristic probabilities for each round

round	1	2	3	4	5	6	7	8	9	10	11	12	13
	B	R_5	R_5	B	R_3	R_3	B	R_5	R_5	B	R_3	R_3	B
active S-box	6	4	0	6	4	0	6	4	0	6	4	0	6
probability($\log_2$)	-16	-34	-34	-50	-68	-68	-84	-102	-102	-118	-136	-136	-152

$$\begin{aligned}(0x08090088, 0, 0, 0) &\xleftarrow{B} (0, 0x00080008, 0x08090088, 0) && 2^{-15}\\ (0, 0) &\xleftarrow{F} (0, 0) && 1\\ (0, 0x00080008) &\xleftarrow{F} (0x08090088, 0) && 2^{-18}.\end{aligned}$$

The differential attack can be applicable to the 13-round SC2000 with 2^{117} data and 2^{162} computational complexity for deriving a part of the extended keys by using the 11-round (2nd to 12th round) differential characteristic with probability 2^{-117} described above and by using two-round elimination technique. Although it is still an open problem whether there exist the effective differential patterns for more rounds or not, we believe that the cipher SC2000 with each of 19-round for 128-bit key and 22-round for other keys is secure against differential attacks.

7 The Security against Linear Attacks

This section describes an evaluation of the security against linear attacks. The symbols used in the evaluation of the security against differential attacks are also used in this session. In the same reason as in the differential searches, we will focus on the iteration of the forms of 3-round linear characteristics.

From the linear distribution table for S_4, (0x2,0x8)-, (0x2,0x9)-, (0x3,0xd)-entry have nonzero probabilities. (See Table 1.) Therefore, the B function in which thirty-two S-boxes S_4 are aligned may possibly have the following input/output linear:

$$(\Gamma d, \Gamma c, 0, \Gamma a) \xleftarrow{B} (0, 0, \Gamma d, \Gamma c),$$

$\Gamma c = mask \wedge \Gamma d$, $\Gamma a \prec \Gamma d$, where $mask$ is the constant value used in the R function. By using this, the following linear characteristic patterns may pass through: (See Figure 5.)

$$\begin{aligned}(\Gamma d, \Gamma c, 0, \Gamma a) &\xleftarrow{B} (0, 0, \Gamma d, \Gamma c)\\ (\Gamma d, \Gamma c) &\xleftarrow{F} (0, \Gamma a)\\ (0, 0) &\xleftarrow{F} (0, 0)\\ (\Gamma d, \Gamma c, 0, \Gamma a) &\xleftarrow{B} (0, 0, \Gamma d, \Gamma c).\end{aligned}$$

As a result of search of linear characteristic patterns of this form by using a computer, we found that the following 3-round linear characteristic is one of the best pattern, whose linear probability is 2^{-34} for 3-round. (See Table 3.)

Table 4. Lower bound of total degrees of algebraic relationals and of numbers of terms in polynomial representations about input bits in n-round input/output bits

# of rounds	1	2	3	4	5	6	7
Total degrees	2	4	8	16	32	64	128

total degree of algebraic relational

# of rounds	1	2	3	4	5	6
# of terms	2^3	2^9	2^{27}	2^{81}	2^{243}	2^{729}

number of term in polynomial

$$(0x08188810, 0x00100010, 0, 0x08100810) \xleftarrow{B} (0, 0, 0x08188810, 0x00100010) \quad 2^{-16}$$
$$(0x08188810, 0x00100010) \xleftarrow{F} (0, 0x08100810) \quad 2^{-18}$$
$$(0, 0) \xleftarrow{F} (0, 0) \quad 1.$$

The linear attack can be applicable for the 13-round SC2000 with 2^{120} data and 2^{180} computational complexity for deriving a part of the extended keys by using the 11-round (2nd to 12th round) linear characteristic with probability 2^{-120} described above by using two-round elimination technique. Although it is still an open problem whether there exist the effective linear patterns for more rounds or not, we believe that the cipher SC2000 with each of 19-round for 128-bit key and 22-round for other keys is secure against linear attacks.

8 The Security against Higher Order Differential Attack and Interpolation Attack

The cipher uses three types of S-boxes: S_4, S_5 and S_6. For all of these S-boxes, nonlinear functions having three or higher degrees are used. In addition, it has been confirmed that algebraic equations for the input/output have two or higher degrees. Therefore, the increase in degrees by two or more per round is unavoidable for the improved higher order differential attacks. Accordingly, it is shown that the minimum degree of algebraic equations for the input/output of n rounds is 2^n or higher. As a result, it is found that the number of required plaintexts exceeds 2^{128} after at most seven rounds.

Here we study a method of finding extended keys by using linear equations, considering Boolean polynomials as vectors of a linear space over $GF(2)$. Assume that the number of terms in a polynomial is estimated by 2^s, where s is the degree of the polynomial (considering that at least two key bits affect). Since the degree of a polynomial increases three per round, the number of terms at the output of the n-th round is estimated as $s = (2^3)^n$ or greater. Therefore, the complexity exceeds 2^{243} for five rounds, and it exceeds 2^{729} for 6 rounds. From the descriptions above, it is assured that improved higher order differential attacks are not successful if the number of rounds is $5 + 5 + 7 = 17$ or greater for the 128-bit key, or $6 + 6 + 7 = 19$ or greater for the 192-bit or 256-bit keys.

The interpolation attack is effective against ciphers that use simple algebraic functions. However, interpolation attacks are often only workable against ciphers whose round functions have very low algebraic degree. The cipher SC2000 is a

combination of Feistel structure and SPN structure, and uses three kind of S-boxes. These combination of the many types of algebraic structures which are used in SC2000 should help increase the degree after a few rounds. Therefore, we believe that SC2000 is secure against interpolation attacks.

9 Study on Collisions of Intermediate-Keys

In the intermediate key generation function, the following properties concerning existence of collisions are satisfied: Let a_i $(i = 0, 1, 2)$ be intermediate keys from an 8-tuple of user keys $k = (k_0, ..., k_7)$ and a'_i $(i = 0, 1, 2)$ from $k' = (k'_0, ..., k'_7)$. Then the following holds: $k_0 \neq k'_0$ or $k_1 \neq k'_1 \Rightarrow a_0 \neq a'_0$ or $a_1 \neq a'_1$.

For intermediate keys b_i, c_i, d_i $(i = 0, 1, 2)$, the same property is satisfied. This property indicates that no collisions occur in the process of generating intermediate keys from the user keys.

Here we prove the property. Let $C_0 = M(S(0))$, $C_1 = M(S(4))$, $\tilde{k}_0 = M(S(k_0))$, $\tilde{k}'_0 = M(S(k'_0))$, $\tilde{k}_1 = M(S(k_1))$,and $\tilde{k}'_1 = M(S(k'_1))$. The i-th bit of variable x is expressed as x_i. From the conditions, we have $(C_0 + \tilde{k}_0) \oplus \tilde{k}_1 = (C_0 + \tilde{k}'_0) \oplus \tilde{k}'_1$, $(C_1 + \tilde{k}_0) \oplus (2\tilde{k}_1) = (C_1 + \tilde{k}'_0) \oplus (2\tilde{k}'_1)$.

Here we use induction on i concerning i-th bit. For the 31st bit (the least significant bit), the following is proven by the latter equation: $C_{1,31} \oplus \tilde{k}_{0,31} = C_{1,31} \oplus \tilde{k}'_{0,31}$, that is, $\tilde{k}_{0,31} = \tilde{k}'_{0,31}$, $C_{0,31} \oplus \tilde{k}_{0,31} \oplus \tilde{k}_{1,31} = C_{0,31} \oplus \tilde{k}'_{0,31} \oplus \tilde{k}'_{1,31}$, that is, $\tilde{k}_{1,31} = \tilde{k}'_{1,31}$. Assume that for any j greater than or equal to $i + 1$, the euqations $\tilde{k}_{0,j} = \tilde{k}'_{0,j}$, $\tilde{k}_{1,j} = \tilde{k}'_{1,j}$ hold. Then for i-th bit, the following equation holds:

$$\begin{aligned} &C_{1,i} \oplus \tilde{k}_{0,i} \oplus \mu(C_{1,31}, ..., C_{1,i+1}, \tilde{k}_{0,31}, ..., \tilde{k}_{0,i+1}) \\ &\quad = C_{1,i} \oplus \tilde{k}'_{0,i} \oplus \mu(C_{1,31}, ..., C_{1,i+1}, \tilde{k}_{0,31}, ..., \tilde{k}_{0,i+1}), \end{aligned}$$

where μ is a Boolean function. From the equation, $\tilde{k}_{0,i} = \tilde{k}'_{0,i}$ is proven, and in the same way $\tilde{k}_{1,i} = \tilde{k}'_{1,i}$ is proven.

10 Study on Some of Weak Keys against Slide Attack

When all values of the 12 intermediate keys a_i, b_i, c_i, d_i, that are generated from user keys are the same, all values of the extended key K_i that are generated in the extended-key generation function become the same. This means that they become weak keys against slide attacks. The following is a study on whether such weak keys exist: We consider the conditions that satisfy the following equation: $a_0 = a_1 = a_2$.

By considering the equations above as 32×2 bitwise Boolean equations, we induced algebraic equations of constraints about $C_0 = M(S(0))$, $C_1 = M(S(4))$, $C_2 = M(S(8))$, by using a computer algebra system. The relational Boolean equations below are parts of such equations. Existence of k_0 and k_1 that satisfies $a_0 = a_1 = a_2$ requires that all of the following Boolean equations must be satisfied:

Table 5. Processing speed in software

(Unit clock cycles / 1 block)

Name	Encode	Decode	ExKey
key size	128 192 256	128 192 256	128 192 256
SPARC	274 323 323	277 317 317	340 393 407
Pentium	383 438 438	403 460 460	427 487 507
Athlon	319 361 361	332 377 377	456 478 476
Alpha	238 298 298	238 298 298	288 327 327

Name	CPU (clock)	Language		Implementation
		En/Decode	ExtKey	(See Section 3.4.)
SPARC	UltraSAPRC II (400MHz)	C	C	(6,10,10,6)
Pentium	PentiumIII (550MHz)	asm.	C	(6,10,10,6)
Athlon	Athlon (900MHz)	asm.	C	(11,10,11)
Alpha	Alpha21264 (500MHz)	C	C	(11,10,11)

Table 6. Performance on Intel 8051 and in JAVA language

Performance in Intel 8051 (128bit key only)

Unit: msec / 1 block

Name	Encode	Decode	ExKey
8051	8.133	8.609	21.666

Size of code: 1597Byte

Size of static data : 1154Byte

Performance of Encryption in JAVA

Unit: clock cycles / 1 block

Name	Encode			Decode			ExKey			Implementation
key length	128	192	256	128	192	256	128	192	256	(See Section 3.4.)
JAVA	3998	4599	4604	4114	4748	4742	14204	15840	15835	(6,5,5,5,5,6)
JAVA	3359	3860	3878	3453	3966	3976	13538	15140	15129	(6,10,10,6)

Binary size: 1200Byte, ROM : 1408Byte (6,5,5,5,5,6) 32790Byte(6,10,10,6)

$$\begin{cases} C_{0,31} \oplus C_{2,31} = 0, \quad C_{0,30} \oplus C_{1,31} \oplus C_{2,30} \oplus C_{2,31} = 0, \\ (C_{1,31} \oplus C_{2,31} \oplus 1)C_{0,29} \oplus (C_{1,31} \oplus C_{2,31} \oplus 1)C_{1,30} \oplus (C_{2,29} \oplus C_{2,30})C_{1,31} \\ \qquad \oplus (C_{2,31} \oplus 1)C_{2,29} \oplus (C_{2,31} \oplus 1)C_{2,30} = 0, \\ \cdots \end{cases}$$

Now, it can be confirmed that the constant parameters (C_0, C_1, C_2) used in the intermediate key generation function, where

$$(C_0, C_1, C_2) \in \{(M(S(x)), M(S(x+4)), M(S(x+8)))\}_{x=0,1,2,3}$$

do not satisfy the system of equations above. Therefore, it can be said that no user key having intermediate keys with the same value exists.

Table 7. Evaluation of hardware implementation

	Key size	Gate size	Throughput
Large†	128-bit	226K	1.26Gbps
Large	128-bit	262K	1.25Gbps
	192-/256-bit	262K	1.08Gbps
Medium	128-bit	63K	964Mbps
	192-/256-bit	63K	844Mbps
Small	128-bit	33K	264Mbps
	192-/256-bit	33K	231Mbps

†: exclusive use for 128-bit key
Verilog-HDL, 0.25μm CMOS ASIC (See [6].)

11 Evaluation of Implementation

In this section, we evaluate the processing speed of the data randomization and key schedule of the cipher.

Table 5 shows the software processing speed for encryption (Encode), decryption (Decode), and key schedule processing (ExKey) on the four CPUs, UltraSPARC II, PentiumIII, Athlon and Alpha21264. Each of the processing speed is measured by clock cycles per one block encryption. For the fast software implementation, the techniques described in Section 3.4 are used. These measurements show the maximum performances which we obtained.

Table 6 shows the software performance of SC2000 on 8-bit CPU measuring by Intel 8051 simulator, and on implementing in JAVA language. For Intel 8051 CPU, Table 6 only shows real timing data (not clock cycles) of the processing functions, since we do not know the ratio between the number of clocks of CPU and the number of cycles of processing and we can not convert from timing data to clock cycles. We note that the performance of the key schedule in our JAVA implementation is measured on the processing speed for swapping all the extended keys without re-construction of the structure of the encryption function.

Table 7 shows our evaluations in hardware implementations. There are three kinds of implementations, Large, Medium and Small. These evaluations of hardware implementation here, Large, Medium and Small, are constructed by giving priorities to the speed of processing, the ratio of the speed to the gate size and the gate size, respectively. The throughputs are measured by bit per second. The total number of gates includes the number of gates for the data randomization and for the key schedule.

References

1. E.Biham, "A Fast New DES Implementation in Software," Fast Software Encryption, 4th International Workshop Proceedings, Springer-Verlag LNCS 1267, 1997, pp.260-272.

2. E.Biham, R.Anderson, and L.Knudsen, "Serpent: A New Block Cipher Proposal," Fast Software Encryption, 5th International Workshop Proceedings, Springer-Verlag LNCS 1372, 1998, pp.222-238.
3. B.Gladman, "Serpent S Boxes as Boolean Functions," http://www.btinternet.com/~brian.gladman/cryptography_technology/serpent/index.html, 2000.
4. J.Daemen, L.Knudsen. V.Rijmen, "The Block Cipher Square," FSE4, LNCS pp.149-165, 1997.
5. J.Daemen, V.Rijmen, "AES Proposal: Rijndael," NIST AES Proposal, Jun 1998.
6. Fujitsu, "CE71 Series," DATA SHEET DS06-20108-1, http://edevice.fujitsu.com/fj/CATALOG/AD00/00-00001/10e-5b-2.html
7. M.Kanda, Y.Takashima, T.Matsumoto, K.Aoki, K.Ohta "A strategy for constructing fast found functions with practical security agaist differential and linear cryptanalysis," Selected Areas in Cryptography, SAC'98, LNCS 1556, pp.264-279, 1998.
8. K.Nyberg and L.R.Knudsen, "Provable Security Against Differential Cryptanalysis," Journal of Cryptology, v.8, n.1, 1995, pp.27-37.
9. D.A.Osvik "Speeding up Serpent," Proceedings of The Third AEC Conference, pp 317-329.
10. B.Schneier, J.Kelsey, D.Whiting, D.Wagner, C.Hall, N.Ferguson, "The Twofish Encryption Algorithm," John Wiley and Sons, Inc. New York, 1999.

Flaws in Differential Cryptanalysis of Skipjack

Louis Granboulan*

École Normale Supérieure
Louis.Granboulan@ens.fr

Abstract. This paper is motivated by some results presented by Knudsen, Robshaw and Wagner at Crypto'99 [3], that described many attacks of reduced versions of Skipjack, some of them being erroneous.
Differential cryptanalysis is based on distinguishers, any attack should prove that the events that triggers the analysis has not the same probability for the cipher than for a random function. In particular, the composition of differential for successive parts of a cipher should be done very carefully to lead to an attack.
This revised version of the paper includes the exact computations of some probabilities and repairs the attack of the first half of Skipjack.

1 What Is Differential Cryptanalysis?

Chosen plaintext attacks. If we have a "black box" containing a symmetric block cipher, we are able to encrypt anything we want. The goal of the attack is to decrypt some given ciphertext, or even better to retrieve the key. A partial success is obtained if we have a distinguisher, i.e. a technique that gives some information about what is in the box (e.g. the algorithm used).

Looking at differences. In order to check the security of a block cipher under chosen plaintext attacks, we can make statistical tests on the output when the input is cleverly chosen. The differential cryptanalysis [2] looks at the difference in the output of the cipher when a pair of input texts with some particular difference (XOR) is enciphered. If the pair of input texts is randomly chosen with their difference following some special distribution of probability, the difference of the outputs may give some information about what is inside.

Building a distinguisher using a differential. More precisely, if we know that, for some keys the input of two different plaintexts with a difference in the subset Δ gives two ciphertexts with a difference in the subset Δ^* with non trivial probability p (this is called a *differential* of probability p, the common notation is $\Delta \rightarrow_p \Delta^*$), then we are able to distinguish two black boxes, one with

* Part of this work has been supported by the CELAR, part of this work has been supported by the Commission of the European Communities through the IST Programme under Contract IST-1999-12324 (NESSIE).

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 328–335, 2002.

the given cipher and the other with a random permutation. Formally, we fix the key and we take probabilities over all pairs (c, c') of cleartexts (or equivalently over all pairs (e, e') of ciphertexts, since the cipher is a permutation). Then $p = \Pr[e \oplus e' \in \Delta^* / c \oplus c' \in \Delta] = \Pr[c \oplus c' \in \Delta / e \oplus e' \in \Delta^*] \frac{\Pr[e \oplus e' \in \Delta^*]}{\Pr[c \oplus c' \in \Delta]}$. Regular differential cryptanalysis is looking for probability close to 1. Impossible cryptanalysis [1] is looking for probability 0. The "trivial probability" is the expected probability p^* that the differential holds for a random permutation. It is the probability that a random value is in Δ^*.

In practice, a regular differential cryptanalysis encrypts n independant random pairs of plaintexts (with $\frac{1}{p} < n < \frac{1}{p^*}$). If one of the pair of ciphertexts has difference in Δ^*, we recognise the cipher not being a random permutation. The probability that the encryption of n pairs of plaintexts produces no pair with difference in Δ^* is less than e^{-np} and the probability that a set of n random pairs of texts contains a pair with difference in Δ^* is less than np^*. If we need better probability of success, we can encrypt more pairs and have a threshold greater than one to decide if the black box is the cipher ; exact probabilities of success can be computed with Chernoff bounds.

Finding weak keys. If the differential holds only for some subset of the keys, the distinguisher allows to detect these keys.

Finding the key of the last rounds with reduced rounds differentials. Most block ciphers are based on a succession of identical rounds, that differ only by the subkey used. The first and last rounds may be different.

If we find a differential (a distinguisher) for the cipher reduced to all but a few last rounds, we can guess with non trivial probability what is the input difference for these few last rounds. Since we exactly know the output, we might be able to find the subkeys used in those rounds. Part of the analysis may be done using the structural properties of how the key bits are used in those rounds and part of the analysis may be done by exhaustive search.

The probability of success is deduced from the gap between the probability p of the differential and the trivial probability p^*. Detailed and practical analysis has been done e.g. in [2].

Composition of differentials. When we can split the cipher in two (ore more) successive ciphers (this is the case with most ciphers, putting the breakpoint between two internal rounds), a very tempting tool is to combine a differential for the first part and one for the second part. This is called a *differential characteristic* and the notation will be $\Delta \to \Delta^\times \to \Delta^*$.

The probability of the differential $\Delta \to \Delta^*$ is greater than or equal to the probability of the differential characteristic $\Delta \to \Delta^\times \to \Delta^*$.

Warning : the probability of the differential characteristic can be unrelated to the probabilities of $\Delta \to \Delta^\times$ and $\Delta^\times \to \Delta^*$. The very simple example below illustrate this fact. $f : (a, b, c) \mapsto (a, b, (a\&b) \oplus c)$. An input difference `100` to f

give an output difference of 100 with probability 1/2 and an output difference of 101 with probability 1/2. However the differential characteristic $\texttt{100} \rightarrow_{1/2} \texttt{100} \rightarrow_{1/2} \texttt{100}$ has probability 1 and the differential characteristic $\texttt{100} \rightarrow_{1/2} \texttt{100} \rightarrow_{1/2} \texttt{101}$ has probability 0.

Markov ciphers [4] have the property that the probability of the differential characteristic $\Delta \rightarrow \Delta^\times \rightarrow \Delta^*$ is equal to the product of the probabilities of $\Delta \rightarrow \Delta^\times$ and $\Delta^\times \rightarrow \Delta^*$.

Finding the key of the first rounds : filtering and counting. Of course, finding the key of the last rounds with a chosen plaintext attack is similar to finding the key of the first rounds with a chosen ciphertext attack. But we limit ourself to the chosen plaintext attacks.

If we have a differential $\Delta^\times \rightarrow \Delta^*$ for the cipher reduced to all but some first rounds, and another differential $\Delta \rightarrow \Delta^\times$ for those rounds, then we use the differential characteristic $\Delta \rightarrow \Delta^\times \rightarrow \Delta^*$ to make an attack as follows.

The cryptanalysis will use (random) pairs of cleartexts having difference in Δ. The filtering selects all pairs of cleartexts such that the ciphertexts have difference in Δ^*. Let p be the probability of $\Delta^\times \rightarrow \Delta^*$, q the probability of $\Delta \rightarrow \Delta^\times$, p^* the probability of $\Delta \rightarrow \Delta^*$. and q^* the probability of $\Delta \rightarrow \Delta^\times \rightarrow \Delta^*$. If q^*/q is substantially greater than p^*, the filtering will increase the probability that a pair of cleartexts have their difference in $\Delta^\times$ after the first rounds. For a Markov cipher $q^* = pq$ and the condition rewrites to $p \gg p^*$. We name it the **filtering condition**. If $p = p^*$, then the highest counter has no reason to be related to the value of the key. The probability of the second differential should not be trivial.

To find the key used in the first part of the cipher, we build a table of counters indexed by all the possible values of this key. For each filtered pair (following $\Delta \rightarrow \Delta^*$), we increase the counters corresponding to all the values that lead to a difference in $\Delta^\times$ after the first rounds. The attack works because the highest counter after looking at all pairs should correspond to the value of the key. This is the **counting hypothesis**. It is not implied by the filtering condition.

2 Example: *Truncated Differentials and Skipjack*

Overview. Knudsen, Robshaw and Wagner presented at Crypto'99 [3] a paper that looks for differentials in Skipjack. They are interested in differentials for the general structure of the algorithm without looking at the details of the "G-boxes" and how the key is used. They propose five attacks of reduced variants of Skipjack:

- Section 4.1 attacks the first 16 round with a composition of a 4-rounds differential and a 12-rounds differential. The differential is used to find the key of the first round.
- Section 4.2 attacks the middle 16 rounds with a distinguisher for the first 12 of them (reduced rounds attack).

- Section 4.3 attacks the last 28 rounds with a composition of a 4-rounds differential and a 24-rounds differential (which is obtained by combining three 8-rounds differentials). The differential is used to find the key of the first round.
- Section 5.1 attacks the middle 24 rounds with a boomerang attack meeting in the middle.
- Section 5.2 is a variant of the previous one, for the middle 25 rounds.

Both attacks of sections 4.1 and 4.3 have the same flaw : the key found by the highest counter is not related to the key of the first round, due to the use of a differential with trivial probability. Then we will look at the boomerang attack of section 5.1, and show that it has a similar flaw.

Notations. We take the notations of [3]. The two different types of rounds of Skipjack are noted τ_A and τ_B. Skipjack is working on blocks of 64 bits splitted in four 16-bits words. The notation for some subset Δ will be $(0, a, a, b)$ for example, indicating that the difference in the first 16-bits word is zero, that the differences in the second and third 16-bits words are equal and non-zero and that the difference in the fourth 16-bits word is non-zero.

Section 4.1. The authors consider the following 16 rounds differential :

$$(a, b, 0, c) \xrightarrow{4\tau_A}_{2^{-32}} (0, d, 0, 0) \xrightarrow{4\tau_A 8\tau_B}_{1} (e, f, g, 0),$$

which is the composition of two differentials and has probability at least 2^{-32}. But we can notice that the differential for the first four rounds is built with the composition $(a, b, 0, c) \xrightarrow{\tau_A}_{2^{-16}} (0, c, b, 0) \xrightarrow{3\tau_A}_{2^{-16}} (0, d, 0, 0)$.

To attack the key of the first round, they use the filtering and counting attack we described before, with the differential :

$$(a, b, 0, c) \xrightarrow{\tau_A}_{2^{-16}} (0, c, b, 0) \xrightarrow{7\tau_A 8\tau_B}_{2^{-16}} (e, f, g, 0)..$$

This attack does not work, because the second differential has trivial probability. The attack could be used to find simultaneously the subkeys of the first four rounds, but they use all the key. It might be corrected by looking at the G-boxes and computing more precisely the probability of $(0, c, b, 0) \xrightarrow{7\tau_A 8\tau_B}$ $(e, f, g, 0)$ and $(a, b, 0, c) \xrightarrow{8\tau_A 8\tau_B} (e, f, g, 0)$. This is done in the annex.

Section 4.3. The authors consider the following 28-rounds differential :

$$(a,b,0,c) \xrightarrow{4\tau_A}_{2^{-16}} (d,e,0,0) \xrightarrow{8\tau_B}_{2^{-16}} (f,g,0,h) \xrightarrow{8\tau_A}_{2^{-32}} (i,i,0,0) \xrightarrow{8\tau_B}_{1} (j,k,l,0),$$

which is the composition of four differentials. We notice that each of them has non trivial probability, with the exception of $(d, e, 0, 0) \xrightarrow{8\tau_B}_{2^{-16}} (f, g, 0, h)$ which cannot distinguish the cipher $8\tau_B$ from a random cipher. The authors want to

use this 28-rounds differential to get some information about the key used in its first round, but it is even not possible if the cipher is restricted to the first twelve rounds of this differential. Indeed, even if someone gives us the value $(f, g, 0, h)$, the differential $(a, b, 0, c) \xrightarrow{4\tau_A}_{2^{-16}} (d, e, 0, 0) \xrightarrow{8\tau_B}_{2^{-16}} (f, g, 0, h)$, which can be rewritten $(a, b, 0, c) \xrightarrow{\tau_A}_{2^{-16}} (0, c, b, 0) \xrightarrow{3\tau_A 8\tau_B}_{2^{-16}} (f, g, 0, h)$, has a second part with trivial probability.

Section 5.1. The authors consider the following 12-rounds forward differential $\Delta = (0, a, 0, 0) \xrightarrow{4\tau_A 8\tau_B}_1 \Delta^* = (c, d, e, 0)$ and the following 12-rounds backward differential $\nabla = (f, 0, 0, 0) \xrightarrow{4\tau_B^{-1} 8\tau_A^{-1}}_1 \nabla^* = (i, h, 0, j)$ that always hold, and build a boomerang attack. Boomerang attacks are chosen-plaintext, adaptive chosen-ciphertext attacks that allow to detect the occurence of some differences in the middle. The attack builds pairs such that $P \oplus P' \in \Delta$, encrypts them to C and C', chooses D and D' such that $C \oplus D \in \nabla$ and $C' \oplus D' \in \nabla$, decrypts them to Q and Q'. The result is a quartet of plaintext/ciphertext pairs that looks good if $Q \oplus Q' \in \Delta$. Good looking quartets have probability 2^{-48} for a random cipher. We will denote $\bar{P}$, $\bar{P}'$, $\bar{Q}$ and $\bar{Q}'$ the results of encryption by half of the cipher. Right quartets have $\bar{P} \oplus \bar{P}' \in \Delta^*$, $\bar{Q} \oplus \bar{Q}' \in \Delta^*$, $\bar{P} \oplus \bar{Q} \in \nabla^*$ and $\bar{P}' \oplus \bar{Q}' \in \nabla^*$.

The authors compute the probability for a quartet of being a right one. The two differentials have probability one, so we have $\bar{P} \oplus \bar{P}' \in \Delta^*$, $\bar{P} \oplus \bar{Q} \in \nabla^*$ and $\bar{P}' \oplus \bar{Q}' \in \nabla^*$ with probability one. The conclusion (footnote in the article) is that $\bar{Q} \oplus \bar{Q}' \in \Delta^*$ happens with probability 2^{-16}. The backward differential $\Delta^* \to \Delta$ having probability 2^{-32} the conclusion of [3] is that right quartets have probability 2^{-48}.

Their error is that they expect two good looking quartets in 2^{48} : one right and one wrong, and that is what distinguishes Skipjack from a random cipher. But with Skipjack (reduced to the middle twenty-four rounds) all good looking quartets are right one, because both differentials $\Delta \to \Delta^*$ and $\nabla \to \nabla^*$ have probability one.

We consider that this mistake is similar to the error in sections 4.1 and 4.3, in the sense that the occurence of some event should not only have interesting probability, but this event should be related to the input of the analysis. If the property $\bar{Q} \oplus \bar{Q}' \in \Delta^*$ had depended on $\bar{P}$ and $\bar{P}'$, the attack could have worked.

References

1. Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *LNCS*, pages 12–23, Prague, May 1999. Springer-Verlag.
2. Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

3. Lars R. Knudsen, M.J.B. Robshaw, and David Wagner. Truncated differentials and skipjack. In Michael Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *LNCS*, pages 165–180, Santa-Barbara, California, August 1999. Springer-Verlag.
4. Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Donald Watts Davies, editor, *Advances in Cryptology, proceedings of Eurocrypt'91*, volume 547 of *LNCS*, pages 17–38, Brighton, UK, April 1991. Springer-Verlag.

A Attacking Skipjack Reduced to the First 16 Rounds

Lemma 1. *We start with N values $(\alpha_i)_{i=1..N}$ taken uniform randomly and independantly from K possible values. We choose randomly two of them α_i and α_j. The expected value for the probablility of $\alpha_i = \alpha_j$ is $\pi = \frac{K+N-1}{NK^2}$. We remark that if $N = K$ then $\pi \simeq \frac{2}{K}$ and if $N = K^2$ then $\pi \simeq \frac{1}{K} + \frac{1}{K^2}$.*
Below we apply this lemma to estimate the probability of $\phi(i) = \phi(j)$ for some random cryptographic function ϕ. [1]

Description of the attack. To attack by fitering and counting the key of the first round of Skipjack reduced to its first 16 rounds, the differential characteristic used in [3, section 4.1] is

$$\Delta = (a, b, 0, c) \quad \xrightarrow{\tau_A} \quad \Delta^{\times} = (0, c, b, 0) \quad \xrightarrow{3\tau_A 8\tau_B} \quad \Delta^{*} = (g, h, f, 0).$$

Let us recall the notations for the probabilities of the differentials. Let p be the probability of $\Delta^{\times} \to \Delta^{*}$, q the probability of $\Delta \to \Delta^{\times}$, p^* the probability of $\Delta \to \Delta^{*}$. and q^* the probability of $\Delta \to \Delta^{\times} \to \Delta^{*}$. The filtering condition is $q^*/q \gg p^*$, which rewrites $p \gg p^*$ if the Markov hypothesis holds.

The Markov hypothesis does not hold in general for Skipjack with truncated differentials, but in that case it holds, because uniform random pairs with difference $(a, b, 0, c)$ such that $(a, b, 0, c) \xrightarrow{\tau_A} (0, c, b, 0)$ give uniform random pairs with difference $(0, c, b, 0)$.

In [3] $p = 2^{-16}$ is computed assuming Markov hypothesis for all truncated differentials and Skipkack, and $p^* = 2^{-16}$ can be computed with the same hypothesis. That result implies that the the attack cannot work. However we see below that the Markov hypothesis does not hold in that case and that $p \simeq 2^{-15}$ and $p^* \simeq 2^{-16}$.

Computation of the probability p of $(0, c, b, 0) \xrightarrow{7\tau_A 8\tau_B} (g, h, f, 0)$. To make an exact computation of p we need to write the result of the encryption of a pair through those 15 rounds of SkipJack. Let G_2 to G_{16} the keyed G functions.

[1] If ϕ is a permutation, this probability is equal to $\frac{1}{K}$. However, if ϕ is obtained by the exclusive XOR of permutations, or similar operations, then the values $\phi(i)$ appear to be taken uniform randomly and the lemma applies.

We also write $\Delta G_i(X)$ the value of $G_i(x_1) \oplus G_i(x_2)$ for some $x_1 \oplus x_2 = X$. If the input pair has difference $(0, c, b, 0)$ then the values of the differences through the 15 rounds are shown in figure 1.

	0	c	b	0	
τ_A	0	0	c	b	
τ_A	b	0	0	c	
τ_A	$c \oplus Z$	Z	0	0	$Z = \Delta G_4(b)$
τ_A	Y	Y	Z	0	$Y = \Delta G_5(c \oplus Z)$
τ_A	X	X	Y	Z	$X = \Delta G_6(Y)$
τ_A	$Z \oplus W$	W	X	Y	$W = \Delta G_7(X)$
τ_A	$Y \oplus V$	V	W	X	$V = \Delta G_8(Z \oplus W)$
τ_B	X	U	Y	W	$U = \Delta G_9(Y \oplus V)$
τ_B	W	T	$U \oplus X$	Y	$T = \Delta G_{10}(X)$
τ_B	Y	S	$W \oplus T$	$U \oplus X$	$S = \Delta G_{11}(W)$
τ_B	$U \oplus X$	R	$Y \oplus S$	$W \oplus T$	$R = \Delta G_{12}(Y)$
τ_B	$W \oplus T$	Q	$U \oplus X \oplus R$	$Y \oplus S$	$Q = \Delta G_{13}(U \oplus X)$
τ_B	$Y \oplus S$	P	$W \oplus T \oplus Q$	$U \oplus X \oplus R$	$P = \Delta G_{14}(W \oplus T)$
τ_B	$U \oplus X \oplus R$	N	$Y \oplus S \oplus P$	$W \oplus T \oplus Q$	$N = \Delta G_{15}(Y \oplus S)$
τ_B	$W \oplus T \oplus Q$	M	$U \oplus X \oplus R \oplus N$	$Y \oplus S \oplus P$	$M = \Delta G_{16}(U \oplus X \oplus R)$

Fig. 1. Differences during $7\tau_A 8\tau_B$ with input difference $(0, c, b, 0)$

Let the input pair be $(\beta, c_1, b_1, \alpha)$, $(\beta, c_2, b_2, \alpha)$ uniform random. Let $\gamma = \alpha \oplus G_2(\beta)$ and $y'_i = \beta \oplus G_5(c_i \oplus G_4(b_i \oplus G_3(\gamma)))$, then the triplet (γ, y'_1, y'_2) is uniform random.

Let $\phi_\gamma : y \mapsto y \oplus G_{11}(G_7(\gamma \oplus G_6(y))) \oplus G_{14}(G_7(\gamma \oplus G_6(y)) \oplus G_{10}(G_6(y)))$. The differential holds if $Y \oplus S \oplus P = 0$, i.e. $\phi_\gamma(y'_1) = \phi_\gamma(y'_2)$.

The function ϕ_γ is a random cryptographic function for our lemma 1 with $N = K = 2^{16}$. The probability of the differential is then estimated to be 2^{-15}.

Computing with exhaustive search for random uniform y'_1, y'_2 the probability of $\phi_\gamma(y'_1) = \phi_\gamma(y'_2)$, we check that the value of p is around 2^{-15}, slightly depending on γ and the key (less than 1% variation).

Computation of the probability p^* of $(a, b, 0, c) \xrightarrow{8\tau_A 8\tau_B} (g, h, f, 0)$. The values of the differences through those 16 rounds are shown in figure 2.

Let the input pair be (a_1, b_1, α, c_1) , (a_2, b_2, α, c_2) uniform random. Let $e'_i = \alpha \oplus G_2(c_i + G_1(a_i))$ and $y'_i = \alpha \oplus G_5(G_1(a_i) \oplus G_4(b_i \oplus G_3(e_i)))$, then the values e'_1, y'_1, e'_2, y'_2 are random uniform.

Let $\psi : (e, y) \mapsto y \oplus G_{11}(G_7(G_3(e) \oplus G_6(e \oplus y))) \oplus G_{14}(G_7(G_3(e) \oplus G_6(e \oplus y)) \oplus G_{10}(G_6(e \oplus y)))$. The differential holds if $\psi(e'_1, y'_1) = \psi(e'_1, y'_2)$. The function ψ is a random cryptographic function for our lemma 1 with $N = 2^{32}$ and $K = 2^{16}$. The probability of the differential is then estimated to be $2^{-16} + 2^{-32}$.

	a	b	0	c	
τ_A	$c \oplus D$	D	b	0	$D = \Delta G_1(a)$
τ_A	E	E	D	b	$E = \Delta G_2(c \oplus D)$
τ_A	$b \oplus F$	F	E	D	$F = \Delta G_3(E)$
τ_A	$D \oplus Z$	Z	F	E	$Z = \Delta G_4(b \oplus F)$
τ_A	$E \oplus Y$	Y	Z	F	$Y = \Delta G_5(D \oplus Z)$
τ_A	$F \oplus X$	X	Y	Z	$X = \Delta G_6(E \oplus Y)$
τ_A	$Z \oplus W$	W	X	Y	$W = \Delta G_7(F \oplus X)$
τ_A	$Y \oplus V$	V	W	X	$V = \Delta G_8(Z \oplus W)$
τ_B	X	U	Y	W	$U = \Delta G_9(Y \oplus V)$
τ_B	W	T	$U \oplus X$	Y	$T = \Delta G_{10}(X)$
τ_B	Y	S	$W \oplus T$	$U \oplus X$	$S = \Delta G_{11}(W)$
τ_B	$U \oplus X$	R	$Y \oplus S$	$W \oplus T$	$R = \Delta G_{12}(Y)$
τ_B	$W \oplus T$	Q	$U \oplus X \oplus R$	$Y \oplus S$	$Q = \Delta G_{13}(U \oplus X)$
τ_B	$Y \oplus S$	P	$W \oplus T \oplus Q$	$U \oplus X \oplus R$	$P = \Delta G_{14}(W \oplus T)$
τ_B	$U \oplus X \oplus R$	N	$Y \oplus S \oplus P$	$W \oplus T \oplus Q$	$N = \Delta G_{15}(Y \oplus S)$
τ_B	$W \oplus T \oplus Q$	M	$U \oplus X \oplus R \oplus N$	$Y \oplus S \oplus P$	$M = \Delta G_{16}(U \oplus X \oplus R)$

Fig. 2. Differences during $8\tau_A 8\tau_B$ with input difference $(a, b, 0, c)$

Computing with semi-exhaustive search for random uniform $(e'_1, y'_1), (e'_2, y'_2)$ the probability of $\psi(e'_1, y'_1) = \psi(e'_1, y'_2)$, we check that the value of p^* is indeed about 2^{-16} with precision better than 2%.

Conclusion. The actual probabilities are $p \simeq 2p^*$. The filtering condition holds and the attack can work. To validate the counting hypothesis and check if the attacks really works, an implementation of the attack is needed.

The attack from [3, section 4.3] of Skipjack reduced to the last 28 rounds is not repaired by similar exact computations.

Efficient Algorithms for Computing Differential Properties of Addition

Helger Lipmaa[1] and Shiho Moriai[2]

[1] Helsinki University of Technology, Department of Computer Science and Engineering
P.O.Box 5400, FI-02015 HUT, Espoo, Finland
helger@tml.hut.fi
[2] NTT Laboratories
1-1 Hikari-no-oka, Yokosuka, 239-0847 Japan
shiho@isl.ntt.co.jp

Abstract. In this paper we systematically study the differential properties of addition modulo 2^n. We derive $\Theta(\log n)$-time algorithms for most of the properties, including differential probability of addition. We also present log-time algorithms for finding good differentials. Despite the apparent simplicity of modular addition, the best known algorithms require naive exhaustive computation. Our results represent a significant improvement over them. In the most extreme case, we present a complexity reduction from $\Omega(2^{4n})$ to $\Theta(\log n)$.

Keywords: Modular addition, differential cryptanalysis, differential probability, impossible differentials, maximum differential probability.

1 Introduction

One of the most successful and influential attacks against block ciphers is Differential Cryptanalysis (DC), introduced by Biham and Shamir in 1991 [BS91a]. For many of the block ciphers proposed since then, provable security against DC (defined by Lai, Massey and Murphy [LMM91] and first implemented by Nyberg and Knudsen [NK95]) has been one of the primary criteria used to confirm their potential quality.

Unfortunately, few approaches to proving security have been really successful. The original approach of [NK95] has been used in designing MISTY and its variant KASUMI (the new 3GPP block cipher standard). Another influential approach has been the "wide trail" strategy proposed by Daemen [Dae95], applied for example in the proposed AES, Rijndael. The main reason for the small number of successful strategies is the complex structure of modern ciphers, which makes exact evaluation of their differential properties infeasible. This has, unfortunately, led to a situation where the security against DC is often evaluated by heuristic methods.

We approach the above problem by using the bottom-up methodology. That is, we evaluate many sophisticated differential properties of one of the most-used "non-trivial" block cipher cornerstones, *addition modulo* 2^n for $n \geq 1$. We hope that this will help to evaluate the differential properties of larger composite cipher parts like the Pseudo-Hadamard Transform, with the entire cipher being the final goal. The algorithms proposed here will enable the advanced cryptanalysis of block ciphers. We hope that our

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 336–350, 2002.

results will facilitate cryptanalysis of such stream ciphers and hash functions that use addition and XOR at the same time.

Importance of Differential Properties of Addition. Originally, DC was considered with respect to XOR, and was generalized to DC with respect to an arbitrary group operation in [LMM91]. In 1992, Berson [Ber92] observed that for many primitive operations, it is significantly more difficult to apply DC with respect to XOR than with respect to addition modulo 2^{32}. Most interestingly, he classified DC of addition modulo 2^n itself, with n sufficiently big, with respect to XOR to be hard to analyze, given the (then) current state of theory.

Until now it has seemed that the problem of evaluating the differential properties of addition with respect to XOR *is* hard. Hereafter, we omit the "with respect to XOR" and take the addition to be always modulo 2^n. The fastest known algorithms for computing the differential probability of addition $\mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma) := \mathbf{P}_{x,y}[(x + y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma]$ is exponential in n. The complexity of the algorithms for the maximum differential probability $\mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta) := \max_{\gamma} \mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$, the double-maximum differential probability $\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha) := \max_{\beta,\gamma} \mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$, and many other differential properties of addition are also exponential in n.

With small n (e.g., $n = 8$ or even with $n = 16$), exponential-in-n computation is feasible, as demonstrated in the cryptanalysis of FEAL by Aoki, Kobayashi and Moriai in [AKM98]. However, this is not the case when $n \geq 32$ as used in the recent 128-bit block ciphers such as MARS, RC6 and Twofish. In practice, if $n \geq 32$, both cipher designers and cryptanalysts have mostly made use of only a few differential properties of addition. (For example, letting x_0 be the least significant bit of x, they often use the property that $\alpha_0 \oplus \beta_0 \oplus \gamma_0 = 0$.) It means that block ciphers that employ both XOR and addition modulo 2^n are hard to evaluate the security against DC due to the lack of theory. This has led to the general concern that mixed use of XOR and modular addition might add more confusion (in Shannon's sense) to a cipher but "none has yet demonstrated to have a clear understanding of how to produce any proof nor convincing arguments of the advantage of such an approach" [Knu99]. One could say that they also add more confusion to the cipher in the layman's sense.

There has been significant ongoing work on evaluating the security of such "confusing" block ciphers against differential attacks. Some of these papers have also somewhat focused on the specific problem of evaluating the differential properties of addition. The full version of [BS91b] treated some differential probabilities of addition modulo 2^n and included a few formulas useful to compute DP^{+}, but did not include any concrete algorithms nor estimations of their complexities. The same is true for many later papers that analyzed ciphers like RC5, SAFER, and IDEA. Miyano [Miy98] studied the simpler case with one addend fixed and derived a linear-time algorithm for computing the corresponding differential probability.

Our Results. We develop a framework that allows the extremely efficient evaluation of many interesting differential properties of modular addition. In particular, most of the algorithms described herein run in time, sublinear in n. Since this would be impossible in the Turing machine model, we chose to use a realistic unit-cost RAM (*Random Access*

Machine) model, which executes basic n-bit operations like Boolean operations and addition modulo 2^n in unit time, as almost all contemporary microprocessors do.

The choice of this model is clearly motivated by the popularity of such microprocessors. Still, for several problems (although sometimes implicitly) we also describe linear-time algorithms that might run faster in hardware. (Moreover, the linear-time algorithms are usually easier to understand and hence serve an educational purpose.) Nevertheless, the RAM model was chosen to be "minimal", such that the described algorithms would be directly usable on as many platforms as possible. On the other hand, we immediately demonstrate the power of this model by describing some useful log-time algorithms (namely, for the Hamming weight, all-one parity and common alternation parity). They become very useful later when we investigate other differential properties. One of them (for the common alternation parity) might be interesting by itself; we have not met this algorithm in the literature.

After describing the model and the necessary tools, we show that DP^+ can be computed in time $\Theta(\log n)$ in the worst-case. The corresponding algorithm has two principal steps. The first step checks in constant-time whether the differential $\delta = (\alpha, \beta \mapsto \gamma)$ is impossible (i.e., whether $\mathrm{DP}^+(\delta) = 0$). The second step, executed only if δ is possible, computes the Hamming weight of an n-bit string in time $\Theta(\log n)$. As a corollary, we prove an open conjecture from [AKM98].

The structure of the described algorithm raises an immediate question of what is the density of the possible differentials. We show that the event $\mathrm{DP}^+(\delta) \neq 0$ occurs with the negligible probability $\frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1}$ (This proves an open conjecture stated in [AKM98]). That is, the density of possible differentials is negligible, so DP^+ can be computed in time $\Theta(1)$ in the average-case. These results can be further used for impossible differential cryptanalysis, since the best previously known general algorithm to find non-trivial impossible differentials was by exhaustive search. Moreover, the high density of impossible differentials makes differential cryptanalysis more efficient; most of the wrong pairs can be filtered out [BS91a,O'C95].

Furthermore, we compute the explicit probabilities $\mathbf{P}_\delta[\mathrm{DP}^+(\delta) = i]$ for any $i,\ 0 \leq i \leq 1$. This helps us to compute the distribution of the random variable $X : \delta \mapsto \mathrm{DP}^+(\delta)$, and to create formulas for the expected value and variance of the random variable X. Based on this knowledge, one can easily compute the probabilities that $\mathbf{P}[X > i]$ for any i.

For the practical success of differential attacks it is not always sufficient to pick a random differential hoping it will be "good" with reasonable probability. It would be nice to find good differentials efficiently in a deterministic way. Both cipher designers and cryptanalysts are especially interested in finding the "optimal" differentials that result in the maximum differential probabilities and therefore in the best possible attacks. For this purpose we describe a log-time algorithm for computing $\mathrm{DP}^+_{\mathsf{max}}(\alpha, \beta)$ and a γ that achieves this probability. Both the structure of the algorithm (which makes use of the all-one parity) and its proof of correctness are nontrivial. We also describe a log-time algorithm that finds a pair (β, γ) that maximizes the double-maximum differential probability $\mathrm{DP}^+_{2\mathsf{max}}(\alpha)$. We show that for many nonzero α-s, $\mathrm{DP}^+_{2\mathsf{max}}(\alpha)$ is very close to one. A summary of some of our results is presented in Table 1.

Table 1. Summary of the efficiency of our main algorithms

	DP^+	$\mathrm{DP}^+_{\mathsf{max}}$	$\mathrm{DP}^+_{\mathsf{2max}}$
Previous result	$\Omega\left(2^{2n}\right)$	$\Omega\left(2^{3n}\right)$	$\Omega\left(2^{4n}\right)$
Our result	$\Theta(\log n)$ (worst-case), $\Theta(1)$ (average)	$\Theta(\log n)$	$\Theta(\log n)$

Road map. We give some preliminaries in Sect. 2. Section 3 describes a unit-cost RAM model, and introduces the reader to several efficient algorithms that are crucial for the later sections. In Sect. 4 we describe a log-time algorithm for DP^+. Section 5 gives formulas for the density of impossible differentials and other statistical properties of DP^+. Algorithms for maximum differential probability and related problems are described in Sect. 6.

2 Preliminaries

Let $\Sigma = \{0,1\}$ be the binary alphabet. For any n-bit string $x \in \Sigma^n$, let $x_i \in \Sigma$ be the i-th coordinate of x (i.e., $x = \sum_{i=0}^{n-1} x_i 2^i$). We always assume that $x_i = 0$ if $i \notin [0, n-1]$. (That is, $x = \sum_{-\infty}^{\infty} x_i 2^i$.)

Let $\oplus$, $\vee$, $\wedge$ and $\neg$ denote n-bit bitwise "XOR", "OR", "AND" and "negation", respectively. Let $x \gg i$ (resp. $x \ll i$) denote the right (resp. the left) shift by i positions (i.e., $x \gg i := \lfloor x/2^i \rfloor$ and $x \ll i := 2^i x \mod 2^n$). Addition is always performed modulo 2^n, if not stated otherwise. For any x, y and z we define $\mathsf{eq}(x,y,z) := (\neg x \oplus y) \wedge (\neg x \oplus z)$ (i.e., $\mathsf{eq}(x,y,z)_i = 1 \iff x_i = y_i = z_i$) and $\mathsf{xor}(x,y,z) := x \oplus y \oplus z$. For any n, let $\mathsf{mask}(n) := 2^n - 1$. For example, $((\neg 0) \ll 1)_0 = 0$.

Addition modulo 2^n. The *carry*, $\mathsf{carry}(x,y) := c \in \Sigma^n$, $x, y \in \Sigma^n$, of addition $x + y$ is defined recursively as follows. First, $c_0 := 0$. Second, $c_{i+1} := (x_i \wedge y_i) \oplus (x_i \wedge c_i) \oplus (y_i \wedge c_i)$, for every $i \geq 0$. Equivalently, $c_{i+1} = 1 \iff x_i + y_i + c_i \geq 2$. (That is, the carry bit c_{i+1} is a function of the *sum* $x_i + y_i + c_i$.) The following is a basic property of addition modulo 2^n.

Property 1. If $(x,y) \in \Sigma^n \times \Sigma^n$, then $x + y = x \oplus y \oplus \mathsf{carry}(x,y)$.

Differential Probability of Addition. We define the *differential* of addition modulo 2^n as a triplet of two input and one output differences, denoted as $(\alpha, \beta \mapsto \gamma)$, where $\alpha, \beta, \gamma \in \Sigma^n$. The *differential probability of addition* is defined as follows:

$$\mathrm{DP}^+(\delta) = \mathrm{DP}^+(\alpha, \beta \mapsto \gamma) := \mathbf{P}_{x,y}[(x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma] \ .$$

That is, $\mathrm{DP}^+(\delta) := \sharp\{x, y : (x+y) \oplus ((x \oplus \alpha) + (y \oplus \beta)) = \gamma\}/2^{2n}$. We say that δ is *impossible* if $\mathrm{DP}^+(\delta) = 0$. Otherwise we say that δ is *possible*. It follows directly from Property 1 that one can rewrite the definition of DP^+ as follows:

Lemma 1. $\mathrm{DP}^+(\alpha, \beta \mapsto \gamma) = \mathbf{P}_{x,y}[\mathsf{carry}(x,y) \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta) = \mathsf{xor}(\alpha, \beta, \gamma)]$.

Probability Theory. Let X be a discrete random variable. Except for a few explicitly mentioned cases, we always deal with uniformly distributed variables. We note that in the *binomial distribution*, $\mathbf{P}[X = k] = p^k(1-p)^{n-k}\binom{n}{k} =: b(k; n, p)$, for some fixed $0 \leq p \leq 1$ and any $k \in \mathbb{Z}_{n+1}$. From the basic axioms of probability, $\sum_{k=0}^{n} b(k; n, p) = 1$. Moreover, the *expectation* $\mathbf{E}[X] = \sum_{k=0}^{n} k \cdot \mathbf{P}[X = k]$ of a binomially distributed random variable X is equal to np, while the *variance* $\mathbf{Var}[X] = \mathbf{E}[X^2] - \mathbf{E}[X]^2$ is equal to $np(1-p)$.

3 RAM Model and Some Useful Algorithms

In the n-bit unit-cost RAM model, some subset of fixed n-bit operations can be executed in constant time. In the current paper, we specify this subset to be a small set of n-bit instructions, all of which are readily available in the vast majority of contemporary microprocessors: Boolean operations, addition, and the constant shifts. We additionally allow unit-cost equality tests and (conditional) jumps. On the other hand, our model does *not* include table look-ups or (say) multiplications. Such a restriction guarantees that algorithms efficient in this model are also efficient on a very broad class of platforms, including FPGA and other hardware. This is further emphasized by the fact that our algorithms need only a few bytes of extra memory and thus a very small circuit size in hardware implementations.

Many algorithms that we derive in the current paper make heavy use of the three non-trivial functions described below. The power of our minimal computational model is stressed by the fact that all three functions can be computed in time $\Theta(\log n)$.

Hamming Weight. The first function is the *Hamming weight function* (also known as the *population count* or, sometimes, as *sideways addition*) $\mathsf{w_h}$: For $x = \sum_{i=0}^{n-1} x_i 2^i$, $\mathsf{w_h}(x) = \sum_{i=0}^{n-1} x_i$, i.e., $\mathsf{w_h}$ counts the "one" bits in an n-bit string. In the unit-cost RAM model, $\mathsf{w_h}(x)$ can be computed in $\Theta(\log n)$ steps. Many textbooks contain (a variation of) the next algorithm that we list here only for the sake of completeness.

INPUT: x
OUTPUT: $\mathsf{w_h}(x)$

1. $x \leftarrow x - ((x \gg 1) \wedge \texttt{0x55555555L})$;
2. $x \leftarrow (x \wedge \texttt{0x33333333L}) + ((x \gg 2) \wedge \texttt{0x33333333L})$;
3. $x \leftarrow (x + (x \gg 4)) \wedge \texttt{0x0F0F0F0FL}$;
4. $x \leftarrow x + (x \gg 8)$;
5. $x \leftarrow (x + (x \gg 16)) \wedge \texttt{0x0000003FL}$;
6. Return x;

Additional time-space trade-offs are possible in calculating the Hamming weight. If $n = cm$, then one can precompute 2^c values of $\mathsf{w_h}(i)$, $0 \leq i < 2^c$, and then find $\mathsf{w_h}(x)$ by doing $m = n/c$ table look-ups. This method is faster than the method described in the previous paragraph if $m \leq \log_2 n$, which is the case if $n = 32$ and $m \in \{8, 16\}$. However, it also requires more memory. While we do not discuss this method hereafter, our implementations use it, since it offers better performance on 32-bit processors.

$$
\begin{aligned}
x &= \texttt{00001000001100110000010101010100}\ ,\\
y &= \texttt{01000000000000010110110001110100}\ ,\\
\mathsf{aop}(x) &= \texttt{00001000001000100000010101010100}\ ,\\
\mathsf{aop}^r(x) &= \texttt{00001000000100010000010101010100}\ ,\\
C(x,y) &= \texttt{00000000000000001000001001001010}\ ,\\
C^r(x,y) &= \texttt{00000000000000010000010010010100}\ .
\end{aligned}
$$

Fig. 1. A pair (x, y) with corresponding values $\mathsf{aop}(x)$, $\mathsf{aop}^r(x)$, $C(x, y)$ and $C^r(x, y)$. Here, for example, $\mathsf{aop}(x)_{27} = 1$ since $1 = x_{27} \neq x_{28}$ and $28 - 27 = 1$ is odd. On the other hand, $C^r(x, y)_4 = 1$ since $x_4 = y_4 \neq x_3 = y_3 \neq x_2 = y_2 \neq x_1 = y_1 = x_0 = y_0$, and $4 - 0$ is even. Since $\ell_5 = 0$, we could have taken also $C(x, y)_5 = 1$

Interestingly, many ancient and modern power architectures have a special machine-level "cryptanalyst's" instruction for $\mathsf{w_h}$ (mostly known as the *population count* instruction): `SADD` on the Mark I (*sic*), `CX`i `X`j on the CDC Cyber series, `A`i `PS`j on the Cray X-MP, `VPCNT` on the NEC SX-4, `CTPOP` on the Alpha 21264, `POPC` on the Ultra SPARC, `POPCNT` on the Intel IA64, etc. In principle, we could incorporate in our model a unit-time population count instruction, then several later presented algorithms would run in constant time. However, since there is no population count instruction on most of the other architectures (especially on the widespread Intel IA32 platform), we have decided not include it in the set of primitive operations. Moreover, the complexity of population count does not significantly influence the (average-case) complexity of the derived algorithms.

All-one and Common Alternation Parity. The second and third functions, important for several derived algorithms (more precisely, they are used in Algorithm 4 and Algorithm 5), are the all-one and common alternation parity of n-bit strings, defined as follows. (Note that while the Hamming weight has very many useful applications in cryptography, the functions defined in this section have never been, as far as we know, used before for any cryptographic or other purpose.)

The *all-one parity* of an n-bit number x is another n-bit number $y = \mathsf{aop}(x)$ s.t. $y_i = 1$ iff the longest sequence of consecutive one-bits $x_i x_{i+1} \ldots x_{i+j} = 11 \ldots 1$ has odd length.

The *common alternation parity* of two n-bit numbers x and y is a function $C(x, y)$ with the next properties: (1) $C(x, y)_i = 1$, if ℓ_i is even and non-zero, (2) $C(x, y)_i = 0$, if ℓ_i is odd, (3) unspecified (either 0 or 1) if $\ell_i = 0$, where ℓ_i is the length of the longest common alternating bit chain $x_i = y_i \neq x_{i+1} = y_{i+1} \neq x_{i+2} = y_{i+2} \ldots \neq x_{i+\ell_i} = y_{i+\ell_i}$, where $i + \ell_i \leq n - 1$. (In both cases, counting starts with one. E.g., if $x_i = y_i$ but $x_{i+1} \neq y_{i+1}$ then $\ell_i = 1$ and $C(x, y)_i = 0$.) W.l.o.g., we will define

$$C(x, y) := \mathsf{aop}(\neg(x \oplus y) \wedge (\neg(x \oplus y) \gg 1) \wedge (x \oplus (x \gg 1)))\ .$$

Algorithm 1 Log-time algorithm for $\mathsf{aop}(x)$

INPUT: $x \in \Sigma^n$, n is a power of 2
OUTPUT: $\mathsf{aop}(x)$

1. $x[1] = x \wedge (x \gg 1)$;
2. For $i \leftarrow 2$ to $\log_2 n - 1$ do $\quad x[i] \leftarrow x[i-1] \wedge (x[i-1] \gg 2^{i-1})$;
3. $y[1] \leftarrow x \wedge \neg x[1]$;
4. For $i \leftarrow 2$ to $\log_2 n$ do $\quad y[i] \leftarrow y[i-1] \vee ((y[i-1] \gg 2^{i-1}) \wedge x[i-1])$;
5. Return $y[\log_2 n]$;

For both the all-one and common alternation parity we will also need their "duals" (denoted as aop^r and C^r), obtained by bit-reversing their arguments. That is, $\mathsf{aop}^r(x, y) = \mathsf{aop}(x', y')$, where $x'_i := x_{n-i}$ and $y'_i := y_{n-i}$. (See Fig. 1.) Note that for every (x, y) and i, $C(x, y)_i = 1 \Rightarrow C(x, y)_{i+1} = C(x, y)_{i-1} = 0$.

Clearly, Algorithm 1 finds the all-one parity of x in time $\Theta(\log n)$. (It is sufficient to note that $x[i]_j = 1$ if and only if the number n_j of ones in the sequence $(x_j = 1, x_{j+1} = 1, \ldots, x_{j+n_j-1} = 1, x_{j+n_j-2} = 0)$ is at least 2^i and $y[i]_j = 1$ iff n_j is an odd number not bigger than 2^j.) Therefore also $C(x, y)$ can be computed in time $\Theta(\log n)$.

4 Log-time Algorithm for Differential Probability of Addition

In this section we say that differential $\delta = (\alpha, \beta \mapsto \gamma)$ is *"good"* if $\mathsf{eq}(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \wedge (\mathsf{xor}(\alpha, \beta, \gamma) \oplus (\alpha \ll 1)) = 0$. Alternatively, δ is not "good" iff for some $i \in [0, n-1]$, $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1} \neq \alpha_i \oplus \beta_i \oplus \gamma_i$. (Remember that $\alpha_{-1} = \beta_{-1} = \gamma_{-1} = 0$.) The next algorithm has a simple linear-time version, suitable for "manual cryptanalysis": (1) Check, whether δ is "good", using the second definition of "goodness"; (2) If δ is "good", count the number of positions $i \neq n-1$, s.t. the triple $(\alpha_i, \beta_i, \gamma_i)$ contains both zeros and ones.

Theorem 1. *Let $\delta = (\alpha, \beta \mapsto \gamma)$ be an arbitrary differential. Algorithm 2 returns $\mathrm{DP}^+(\delta)$ in time $\Theta(\log n)$. More precisely, it works in time $\Theta(1) + t$, where t is the time it takes to compute $\mathsf{w_h}$.*

Algorithm 2 Log-time algorithm for DP^+

INPUT: $\delta = (\alpha, \beta \mapsto \gamma)$
OUTPUT: $\mathrm{DP}^+(\delta)$

1. If $\mathsf{eq}(\alpha \ll 1, \beta \ll 1, \gamma \ll 1) \wedge (\mathsf{xor}(\alpha, \beta, \gamma) \oplus (\beta \ll 1)) \neq 0$ then return 0;
2. Return $2^{-\mathsf{w_h}(\neg \mathsf{eq}(\alpha, \beta, \gamma) \wedge \mathsf{mask}(n-1))}$;

Rest of this subsection consists of a step-by-step proof of this result, where we use the Lemma 1, i.e., that $\mathrm{DP}^+(\delta) = \mathbf{P}_{x,y}[\mathsf{carry}(x, y) \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta) = \mathsf{xor}(\alpha, \beta, \gamma)$.

We first state and prove two auxiliary lemmas. After that we show how Theorem 1 follows from them, and present two corollaries.

Lemma 2. *Let $L(x)$ be a mapping, such that $L(0) = 0$, $L(1) = L(2) = \frac{1}{2}$ and $L(3) = 1$. Let $\alpha, \beta \in \Sigma^n$. Then $\mathbf{P}_{x,y}[\mathsf{carry}(x,y)_{i+1} \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_{i+1} = 1 | \alpha_i + \beta_i + \Delta c_i = j] = L(j)$.*

Proof. We denote $c = \mathsf{carry}(x,y)$ and $c^* = \mathsf{carry}(x \oplus \alpha, y \oplus \beta)$, where x and y are understood from the context. Let also $\Delta c = c \oplus c^*$. By the definition of carry, $\Delta c_{i+1} = (x_i \wedge y_i) \oplus (x_i \wedge c_i) \oplus (y_i \wedge c_i) \oplus ((x \oplus \alpha)_i \wedge (y \oplus \beta)_i) \oplus ((x \oplus \alpha)_i \wedge c_i^*) \oplus ((y \oplus \beta)_i \wedge c_i^*)$. This formula for Δc_{i+1} is symmetric in the three pairs (x_i, α_i), (y_i, β_i) and $(c_i, \Delta c_i)$. Hence, the function $f(\alpha_i, \beta_i, \Delta c_i) := \mathbf{P}_{x_i,y_i,c_i}[\Delta c_{i+1} = 1]$ is symmetric, and therefore f is a function of $\alpha_i + \beta_i + \Delta c_i$, $f(j) = \mathbf{P}_{x_i,y_i,c_i}[\Delta c_{i+1} = 1 | \alpha_i + \beta_i + \Delta c_i = j]$. One can now prove that $\mathbf{P}_{x_i,y_i}[\Delta c_{i+1} = 1 | \alpha_i + \beta_i + \Delta c_i = j] = L(j)$ for any $0 \le j \le 3$, and for any value of $c_i \in \{0,1\}$. For example, $\mathbf{P}_{x_i,y_i}[\Delta c_{i+1} = 1 | \alpha_i + \beta_i + \Delta c_i = 1] = \mathbf{P}_{x_i,y_i}[\Delta c_{i+1} = 1 | (\alpha_i, \beta_i, \Delta c_i) = (0,0,1)] = \mathbf{P}_{x_i,y_i}[(x_i \wedge c_i) \oplus (y_i \wedge c_i) \oplus (x_i \wedge \neg c_i) \oplus (x_i \wedge \neg c_i) = 1] = \mathbf{P}_{x_i,y_i}[x_i = y_i] = \frac{1}{2}$. The claim follows since $\mathbf{P}_{x,y}[\Delta c_{i+1}] = \mathbf{P}_{x_i,y_i}[\Delta c_{i+1}]$. □

Lemma 3. *1) Every possible differential is "good".*
2) Let $\delta = (\alpha, \beta \mapsto \gamma)$ be "good". If $i \in [0, n-1]$, then $\mathbf{P}_{x,y}[\mathsf{carry}(x,y)_i \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_i = 1 | \alpha_{i-1} + \beta_{i-1} + \gamma_{i-1} = j] = L(j)$. In particular, $\mathbf{P}_{x,y}[\mathsf{carry}(x,y)_0 \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_0 = 0] = 1$.

Proof. 1) Let δ be possible but not "good". By Lemma 1, there exists an i and a pair (x,y), s.t. $\mathsf{carry}(x,y)_{i+1} \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_{i+1} = \mathsf{xor}(\alpha, \beta, \gamma)_{i+1} \neq \alpha_i = \beta_i = \gamma_i$. Note that then $\mathsf{xor}(\alpha, \beta, \gamma)_i = \gamma_i$. But by Lemma 2, $\mathbf{P}_{x_i,y_i}[\mathsf{carry}(x,y)_{i+1} \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_{i+1} \neq \gamma_i | \alpha_i = \beta_i = \gamma_i] = 0$, which is a contradiction.

2) Let δ be "good". We prove the theorem by induction on i, by simultaneously proving the induction invariant $\mathbf{P}_{x,y}[\mathsf{carry}(x,y) \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta) = \mathsf{xor}(\alpha, \beta, \gamma) \pmod{2^i}] > 0$. Base ($i = 0$). Straightforward from Property 1 and the definition of a "good" differential. Step ($i + 1 > 0$). We assume that the invariant is true for i. In particular, there exists a pair (x,y), s.t. $\Delta c_{i-1} = \mathsf{xor}(\alpha, \beta, \gamma)_{i-1}$, where $\Delta c = \mathsf{carry}(x,y) \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)$. Then, by Lemma 2, $L(j) = \mathbf{P}_{x,y}[\Delta c_i = 1 | \alpha_{i-1} + \beta_{i-1} + \Delta c_{i-1} = j] = \mathbf{P}_{x,y}[\Delta c_i = 1 | \alpha_{i-1} + \beta_{i-1} + \mathsf{xor}(\alpha, \beta, \gamma)_{i-1} = j] = \mathbf{P}_{x,y}[\Delta c_i = 1 | \alpha_{i-1} + \beta_{i-1} + \gamma_{i-1} = j]$, where the last equation follows from the easily verifiable equality $L(a_1 + a_2 + a_3) = L(a_1 + a_2 + \mathsf{xor}(a_1, a_2, a_3))$, for every $a_1, a_2, a_3 \in \Sigma$. This proves the theorem claim for i. The invariant for i, $\Delta c_i = \mathsf{xor}(\alpha, \beta, \gamma)_i$, follows from that and the "goodness" of δ. □

Proof (Theorem 1). First, δ is "good" iff it is possible. (The "if" part follows from the first claim of Lemma 3. The "only if" part follows from the second claim of Lemma 3 and the definition of a "good" differential.) Let δ be possible. Then, by Lemma 1, $\mathrm{DP}^+(\delta) = \prod_{i=0}^{n-2} \mathbf{P}_{x,y}[\mathsf{carry}(x,y)_i \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_i = \mathsf{xor}(\alpha, \beta, \gamma)_i]$. By Lemma 2, $\mathbf{P}_{x,y}[\mathsf{carry}(x,y)_i \oplus \mathsf{carry}(x \oplus \alpha, y \oplus \beta)_i = \mathsf{xor}(\alpha, \beta, \gamma)_i]$ is either 1 or $\frac{1}{2}$, depending on whether $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}$ or not. (This probability cannot be 0,

since δ is possible and hence "good".) Therefore, $\mathrm{DP}^+(\delta) = 2^{-\sum_{i=0}^{n-2} \neg\mathsf{eq}(\alpha,\beta,\gamma)_i} = 2^{-\mathsf{w_h}(\neg\mathsf{eq}(\alpha,\beta,\gamma)\wedge\mathsf{mask}(n-1))}$, as required. Finally, the only non-constant time computation is that of Hamming weight. □

Note that *technically*, for Algorithm 2 to be log-time it would have to return (say) -1 if the differential is impossible, or $\log_2 \mathrm{DP}^+(\delta)$, if it is not. (The other valid possibility would be to include data-dependent shifts in the set of unit-cost operations.)

The next two corollaries follow straightforwardly from Algorithm 2

Corollary 1. DP^+ *is symmetric in its arguments. That is, for an arbitrary triple* (α, β, γ), $\mathrm{DP}^+(\alpha, \beta \mapsto \gamma) = \mathrm{DP}^+(\beta, \alpha \mapsto \gamma) = \mathrm{DP}^+(\alpha, \gamma \mapsto \beta)$. *Therefore, in particular,* $\max_\alpha \mathrm{DP}^+(\alpha, \beta \mapsto \gamma) = \max_\beta \mathrm{DP}^+(\alpha, \beta \mapsto \gamma) = \max_\gamma \mathrm{DP}^+(\alpha, \beta \mapsto \gamma)$.

Corollary 2. *1) [Conjecture 1, [AKM98].] Let* $\alpha + \beta = \alpha' + \beta'$ *and* $\alpha \oplus \beta = \alpha' \oplus \beta'$. *Then for every* γ, $\mathrm{DP}^+(\alpha, \beta \mapsto \gamma) = \mathrm{DP}^+(\alpha', \beta' \mapsto \gamma)$. *2) For every* α, β, γ, $\mathrm{DP}^+(\alpha, \beta \mapsto \gamma) = \mathrm{DP}^+(\alpha \wedge \beta, \alpha \vee \beta \mapsto \gamma)$.

Proof. We say that (α, β) and (α', β') are *equivalent*, if $\{\alpha_i, \beta_i\} = \{\alpha'_i, \beta'_i\}$ for $i < n-1$, and $\alpha_{n-1} \oplus \beta_{n-1} = \alpha'_{n-1} \oplus \beta_{n-1}$. If (α, β) and (α', β') are equivalent then $\mathrm{DP}^+(\alpha, \beta \mapsto \gamma) = \mathrm{DP}^+(\alpha', \beta' \mapsto \gamma)$ by the structure of Algorithm 2.

1) The corresponding carries $c = \mathsf{carry}(\alpha, \beta)$ and $c' = \mathsf{carry}(\alpha', \beta')$ are equal, since $c = (\alpha \oplus \beta) \oplus (\alpha + \beta) = (\alpha' \oplus \beta') \oplus (\alpha' + \beta') = c'$. Therefore, $\alpha + \beta = \alpha' + \beta'$ and $\alpha \oplus \beta = \alpha' \oplus \beta'$ iff (α, β) and (α', β') are equivalnet.

2) The second claim is straightforward, since (α, β) and $(\alpha \wedge \beta, \alpha \vee \beta)$ are equivalent for any α and β. □

Note that a pair (x, y) is equivalent to $2^{1+\mathsf{w_h}((x\oplus y)\wedge\mathsf{mask}(n-1))}$ different pairs (x^*, y^*). In [DGV93, Sect. 2.3] it was briefly mentioned that the number of such pairs is not more than $2^{\mathsf{w_h}(x\oplus y)}$; this result was used to cryptanalyse IDEA. The second claim carries unexpected connotations with the well known fact that $\alpha + \alpha' = (\alpha \wedge \alpha') + (\alpha \vee \alpha')$.

5 Statistical Properties of Differential Probability

Note that Algorithm 2 has two principal steps. The first step is a constant-time check of whether the differential $\delta = (\alpha, \beta \mapsto \gamma)$ is impossible (i.e., whether $\mathrm{DP}^+(\delta) = 0$). The second step, executed only if δ is possible, computes in log-time the Hamming weight of an n-bit string. The structure of this algorithm raises an immediate question of what is the density $\mathbf{P}_\delta[\mathrm{DP}^+(\delta) \neq 0]$ of the possible differentials, since its average-case complexity (where the average is taken over uniformly and random chosen differentials δ) is $\Theta(\mathbf{P}_\delta[\mathrm{DP}^+(\delta) = 0] + \mathbf{P}_\delta[\mathrm{DP}^+(\delta) \neq 0] \cdot \log n)$. This is one (but certainly not the only or the most important) motivation for the current section.

Let $X : \delta \mapsto \mathrm{DP}^+(\delta)$ be a uniformly random variable. We next calculate the exact probabilities $\mathbf{P}[X = i]$ for any i. From the results we can directly derive the distribution of X. Knowing the distribution, one can, by using standard probabilistic tools, calculate the values of many other interesting probabilistic properties like the probabilities $\mathbf{P}[X > i]$ for any i.

Theorem 2. *1) [Conjecture 2, [AKM98].]* $\mathbf{P}[X \neq 0] = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1}$.
2) Let $0 \leq k < n$. *Then* $\mathbf{P}[X = 2^{-k}] = 2^{2+k-3n} \cdot 3^k \cdot \binom{n-1}{k} = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1} \cdot b\left(k; n-1, \frac{6}{7}\right)$.

Proof. Let $\delta = (\alpha, \beta \mapsto \gamma)$ be an arbitrary differential and let $e = \mathsf{eq}(\alpha, \beta, \gamma)$, $e' = \mathsf{eq}(\alpha \ll 1, \beta \ll 1, \gamma \ll 1)$ and $x = \mathsf{xor}(\alpha, \beta, \gamma) \oplus (\alpha \ll 1))$ be convenient shorthands. Since α, β and γ are mutually independent, e and x (and also e' and x) are pairwise independent.

1) From Theorem 1, $\mathbf{P}[X \neq 0] = \mathbf{P}_\delta[e' \wedge x = 0] = \prod_{i=0}^{n-1}(1 - \mathbf{P}_\delta[e'_i = 1, x_i = 1]) = \prod_{i=0}^{n-1}(1 - \mathbf{P}_\delta[e'_i = 1] \cdot \mathbf{P}_\delta[x_i = 1]) = \left(1 - 1 \cdot \frac{1}{2}\right) \cdot \prod_{i=1}^{n-1}\left(1 - \frac{1}{4} \cdot \frac{1}{2}\right) = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1}$.

2) Let $m = \mathsf{mask}(n-1)$. First, clearly, for any $0 \leq k < n$, $\mathbf{P}_\delta[\mathsf{w_h}(e) = k] = \mathbf{P}_{\alpha,\beta,\gamma \in \Sigma^n}[\mathsf{w_h}(e) = k] = \left(\frac{1}{4}\right)^k \cdot \left(\frac{3}{4}\right)^{n-k} \cdot \binom{n}{k} = b\left(k; n, \frac{1}{4}\right)$ and therefore $\mathbf{P}_\delta[\mathsf{w_h}(\neg e \wedge m) = k] = b\left(n-1-k; n-1, \frac{1}{4}\right) = \left(\frac{1}{4}\right)^{n-1-k} \cdot \left(\frac{3}{4}\right)^k \cdot \binom{n-1}{k} = 2^{2-2n} \cdot 3^k \cdot \binom{n-1}{k}$.

Let A denote the event $\mathsf{w_h}(e \wedge m) = n-1-k$ and let B denote the event $e' \wedge x = 0$. Let B_i be the event $e'_i \wedge x_i = 0$. According to Algorithm 2, $\mathbf{P}[X = 2^{-k}] = \mathbf{P}_\delta[A, B] = \mathbf{P}_\delta[A] \cdot \mathbf{P}_\delta[B|A] = \mathbf{P}_\delta[A] \cdot \prod_{i=0}^{n-1} \mathbf{P}_\delta[B_i|A] = \frac{1}{2} \cdot \mathbf{P}_\delta[A] \cdot \prod_{i=1}^{n-1} \mathbf{P}_\delta[B_i|A]$, where we used the fact that $e'_0 = 1$.

Now, if $i > 0$ then $\mathbf{P}_\delta[B_i|e'_i = 1] = \mathbf{P}_\delta[x_i = 0] = \frac{1}{2}$, while $\mathbf{P}_\delta[B_i = 0|e'_i = 0] = 1$. Moreover, $e'_i = e_{i-1}$. Therefore, $\prod_{i=0}^{n-1} \mathbf{P}_\delta[B_i|A] = \left(\frac{1}{2}\right)^{n-1-k}$, and hence $\mathbf{P}[X = 2^{-k}] = \frac{1}{2} \cdot \mathbf{P}_\delta[A] \cdot \prod_{i=0}^{n-1} \mathbf{P}_\delta[B_i|A] = \frac{1}{2} \cdot b\left(n-1-k; n-1, \frac{1}{4}\right) \cdot \left(\frac{1}{2}\right)^{n-1-k} = \frac{1}{2} \cdot 2^{2-2n} \cdot 3^k \cdot \binom{n-1}{k} \cdot 2^{1+k-n} = 2^{2+k-3n} \cdot 3^k \cdot \binom{n-1}{k} = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1} \cdot b\left(k; n-1, \frac{6}{7}\right)$. □

Corollary 3. *Algorithm 2 has average-case complexity* $\Theta(1)$.

As another corollary, $X = X_0 + X_1$, where $X_1, X_2 : \delta \mapsto \mathrm{DP}^+(\delta)$ are two random variables. X_0 has domain $\mathcal{D}(X_0) = \{\delta \in \Sigma^{3n} : \mathrm{DP}^+(\delta) = 0\}$, while X_1 has the complementary domain $\mathcal{D}(X_1) = \{\delta \in \Sigma^{3n} : \mathrm{DP}^+(\delta) \neq 0\}$. Moreover, X_0 has constant distribution (since $\mathbf{P}[X_0 = 0] = 1$), while the random variable $-\log_2 X_1$ has binomial distribution with $p = \frac{6}{7}$. Knowledge of the distribution helps to find further properties of DP^+ (e.g., the probabilities that $\mathrm{DP}^+(\delta) > 2^{-k}$) by using standard methods from probability theory.

One can double-check the correctness of Theorem 2 by verifying that $\frac{1}{2} \cdot (\frac{7}{8})^{n-1} \cdot \sum_{k=0}^{n-1} b\left(k; n-1, \frac{6}{7}\right) = \sum_{k=0}^{n-1} \mathbf{P}[X = 2^{-k}] = \mathbf{P}[X \neq 0] = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1}$. Moreover, clearly $\mathbf{P}[X = 2^{-k}] = |\mathcal{D}(X_0)| \cdot \mathbf{P}[X_0 = 2^{-k}] + |\mathcal{D}(X_1)| \cdot \mathbf{P}[X_1 = 2^{-k}] = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1} \cdot b\left(k; n-1, \frac{6}{7}\right)$, which agrees with Theorem 2.

We next compute the variance of X. Clearly, $\mathbf{E}[X] = \sum_{k=0}^{n-1} 2^{-k} \mathbf{P}[X = 2^{-k}] = 2^{-n}$, and therefore $\mathbf{E}[X]^2 = 2^{-2n}$. Next, by using Theorem 2 and the basic properties of the binomial distribution, $\mathbf{E}[X^2] = 0 \cdot \mathbf{P}[X^2 = 0] + \sum_{k=0}^{n-1} 2^{-2k} \cdot \mathbf{P}[X^2 = 2^{-2k}] = \frac{1}{2} \cdot \left(\frac{7}{8}\right)^{n-1} \cdot \sum_{k=0}^{n-1} 2^{-2k} \cdot b\left(k; n-1, \frac{6}{7}\right) = \frac{1}{2} \cdot \left(\frac{5}{16}\right)^{n-1} \cdot \sum_{i=0}^{n-1} b\left(k; n-1, \frac{3}{5}\right) = \frac{1}{2} \cdot \left(\frac{5}{16}\right)^{n-1}$. Therefore, $\mathbf{Var}[X] = \frac{1}{2} \cdot \left(\frac{5}{16}\right)^{n-1} - 2^{-2n} = \frac{1}{2} \cdot \left(\left(\frac{5}{16}\right)^{n-1} - \left(\frac{4}{16}\right)^{n-1}\right)$.

Note that the density of possible differentials $\mathbf{P}[X \neq 0]$ is exponentially small in n. This can be contrasted with a result of O'Connor [O'C95] that a randomly selected n-bit

Algorithm 3 Algorithm that finds all γ-s, s.t. $\mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma) = \mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta)$

INPUT: (α, β)
OUTPUT: All (α, β)-optimal output differences γ

1. $\gamma_0 \leftarrow \alpha_0 \oplus \beta_0$;
2. $p \leftarrow C(\alpha, \beta)$;
3. For $i \leftarrow 1$ to $n-1$ do
   If $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}$ then $\gamma_i \leftarrow \alpha_i \oplus \beta_i \oplus \alpha_{i-1}$
   else if $i = n-1$ or $\alpha_i \neq \beta_i$ or $p_i = 1$ then $\gamma_i \leftarrow \{0, 1\}$
   else $\gamma_i \leftarrow \alpha_i$;
4. Return γ.

permutation has a fraction of $1 - e^{-1/2} \approx 0.4$ impossible differentials, independently of the choice of n. Moreover, a randomly selected n-bit composite permutation [O'C93], controlled by an n-bit string, has a negligible fraction $\approx 2^{3n}/e^{2^n - 1}$ of impossible differentials.

6 Algorithms for Finding Good Differentials of Addition

The last section described methods for computing the probability that a randomly picked differential δ has high differential probability. While this alone might give rise to successful differential attacks, it would be nice to have an efficient deterministic algorithm for finding differentials with high differential probability. This section gives some relevant algorithms for this.

6.1 Linear-Time Algorithm for $\mathrm{DP}^{+}_{\mathsf{max}}$

In this subsection, we will describe an algorithm that, given an input difference (α, β), finds all output differences γ, for which $\mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$ is equal to the *maximum differential probability of addition*, $\mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta) := \max_{\gamma} \mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$. (By Corollary 1, we would get exactly the same result when maximizing the differential probability under α or β.) We say that such γ is *(α, β)-optimal*. Note that when an (α, β)-optimal γ is known, the maximum differential probability can be found by applying Algorithm 2 to $\delta = (\alpha, \beta \mapsto \gamma)$. Moreover, similar algorithms can be used to find "near-optimal" γ-s, where $\log_2 \mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma)$ is only slightly smaller than $\log_2 \mathrm{DP}^{+}_{\mathsf{max}}(\alpha, \beta)$.

Theorem 3. *Algorithm 3 returns all (α, β)-optimal output differences γ.*

Proof (Sketch). First, we say that position i is *bad* if $\mathsf{eq}(\alpha, \beta, \gamma)_i = 0$. According to Theorem 1, γ is (α, β)-optimal if it is chosen so that (1) for every $i \geq 0$, if $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 1$ then $\mathsf{xor}(\alpha_i, \beta_i, \gamma_i) = \alpha_{i-1}$, and (2) the number of bad positions i is the least among all such output differences γ', for which $(\alpha, \beta \mapsto \gamma')$ is possible. For achieving (1) we must first fix $\gamma_0 \leftarrow \alpha_0 \oplus \beta_0$, and after that recursively guarantee that γ_i obtains the predicted value whenever $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}$.

Algorithm 4 A log-time algorithm that finds an (α, β)-optimal γ

INPUT: (α, β)
OUTPUT: An (α, β)-optimal γ

1. $r \leftarrow \alpha \wedge 1$;
2. $e \leftarrow \neg(\alpha \oplus \beta) \wedge \neg r$;
3. $a \leftarrow e \wedge (e \ll 1) \wedge (\alpha \oplus (\alpha \ll 1))$;
4. $p \leftarrow \mathsf{aop}^r(a)$;
5. $a \leftarrow (a \vee (a \gg 1)) \wedge \neg r$;
6. $b \leftarrow (a \vee e) \ll 1$;
7. $\gamma \leftarrow ((\alpha \oplus p) \wedge a) \vee ((\alpha \oplus \beta \oplus (\alpha \ll 1)) \wedge \neg a \wedge b) \vee (\alpha \wedge \neg a \wedge \neg b)$;
8. $\gamma \leftarrow (\gamma \wedge \neg 1) \vee ((\alpha \oplus \beta) \wedge 1)$;
9. Return γ.

This, and minimizing the number of bad i-s can be done recursively for every $0 \leq i \leq n-1$, starting from $i = 0$. If $\alpha_i \neq \beta_i$ then i is bad independently of the value of $\gamma_i \in \{0, 1\}$. Moreover, either choice of γ places no restriction on choosing γ_{i+1}. This means that we can assign either $\gamma_i \leftarrow 0$ or $\gamma_i \leftarrow 1$.

The situation is more complicated if $\alpha_i = \beta_i$. Intuitively, if $\ell_i = 2k > 0$ is even, then the choice $\gamma_i \leftarrow \alpha_i$ (as compared to the choice $\gamma_i \leftarrow \neg\alpha_i$) will result in k bad positions $(i, i+2, \ldots, i+2k-2)$ instead of k bad positions $(i+1, i+3, \ldots, i+2k-1)$. Thus these two choices are equal. On the other hand, if $\ell_i = 2k+1 > 0$, then the choice $\gamma_i \leftarrow \alpha_i$ would result in k bad positions compared to $k+1$ when $\gamma_i \leftarrow \alpha_i$, and hence is to be preferred over the second one. We leave the full details of the proof to the reader. □

A linear-time algorithm that finds one (α, β)-optimal γ can be derived from Algorithm 3 straightforwardly, by assigning $\gamma_i \leftarrow \alpha_i$ whenever $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 0$.

As an example, let us look at the case $n = 16$, $\alpha = \texttt{0x5254}$ and $\beta = \texttt{0x1A45}$. Then $C(\alpha, \beta) = \texttt{0x1244}$, and by Algorithm 3 the set of (α, β)-optimal values is equal to $2^{15}\Sigma + 2^{12}P + 2^{11}\Sigma + 2^4\Sigma + 1$, where $P = \{0, 3, 7\}$, and $\Sigma = \{0, 1\}$, as always. Therefore, for example, $\mathrm{DP}^+_{\mathsf{max}}(\texttt{0x5254}, \texttt{0x1A45}) = \mathrm{DP}^+(\texttt{0x5254}, \texttt{0x1A45} \mapsto \texttt{0x7011}) = 2^{-8}$.

6.2 Log-time Algorithm for $\mathrm{DP}^+_{\mathsf{max}}$

For a log-time algorithm we need a somewhat different approach, since in Algorithm 3 the value of γ_i depends on that of γ_{i-1}. However, luckily for us, γ_i only depends on γ_{i-1} if $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 1$, and as seen from the proof of Theorem 3, in many cases we can choose the output difference γ_{i-1} so that $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 0$!

Moreover, the positions where $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 1$ must hold are easily detected. Namely (see Algorithm 3), if (1) $i = 0$ and $\alpha_i = \beta_i = 0$, or (2) $i > 0$ and $\alpha_i = \beta_i$ but $p_i = 0$. Accordingly, we can replace the condition $\mathsf{eq}(\alpha, \beta, \gamma)_{i-1} = 1$ with the condition $\neg(\alpha_i \oplus \beta_i) \wedge p_i$, and take additional care if i is small. By noting how the values p_i are computed, one can prove that

Algorithm 5 A log-time algorithm for $\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha)$

INPUT: α
OUTPUT: $\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha)$

1. Return $2^{-\mathsf{w_h}(C^r(\alpha,\alpha)\wedge\mathsf{mask}(n-1))}$.

Theorem 4. *Algorithm 4 finds an (α,β)-optimal γ.*

Proof (Sketch, many details omitted). First, the value of p computed in the step 4 is "approximately" equal to $C^r(\alpha,\beta)$, with some additional care taken about the lowest bits. Let ℓ^r be the bit-reverse of ℓ (i.e., ℓ^r_i is equal to the length of longest common alternating chain $\alpha_i=\beta_i\neq\alpha_{i-1}=\beta_{i-1}\neq\ldots$.). Step 7 computes γ_i (again, "approximately") as (1) $\gamma_i\leftarrow\alpha_i\oplus p_i$, if $\alpha_i=\beta_i$, (2) $\gamma_i\leftarrow\alpha_i\oplus\beta_i\oplus\alpha_{i-1}$ if $\alpha_i\neq\beta_i$ but $\mathsf{eq}(\alpha,\beta,\gamma)_{i-1}=1$ and (3) $\gamma_i\leftarrow\alpha_i$ if $\alpha_i\neq\beta_i$ and $\mathsf{eq}(\alpha,\beta,\gamma)_{i-1}=0$. Since the two last cases are sound, according to Algorithm 3, we are now left to prove that the first choice makes γ optimal.

But this choice means that in every maximum-length common alternating bit chain $\alpha_i=\beta_i\neq\alpha_{i-1}=\beta_{i-1}\neq\ldots\neq\alpha_{i-\ell^r_i+1}=\beta_{i-\ell^r_i+1}$, Algorithm 4 chooses all bits γ_j, $j\in[i-\ell^r_i+1,i]$, to be equal to $\alpha_{i-\ell^r_i+1}=\beta_{i-\ell^r_i+1}$. By approximately the same arguments as in the proof of Theorem 3, this choice gives rise to $\lfloor\ell_i/2\rfloor$ bad bit positions in the fragment $[i-\ell^r_i+1,i]$; every other choice of bits γ_j would result in at least as many bad positions. Moreover, since $\neg(\alpha_{i+1}=\beta_{i+1}\neq\alpha_i=\beta_i)$, it has to be the case that either $\alpha_{i+1}\neq\beta_{i+1}$ or $\alpha_{i+1}=\beta_{i+1}=\alpha_i=\beta_i$. In the first case, both choices of γ_i make $i+1$ bad. In the second case we must later take $\gamma_{i+1}\leftarrow\alpha_{i+1}$, which makes $i+1$ good, and enables us to start the next fragment from the position $i+1$. (Intuitively, this last fact is also the reason why aop^r is here preferred over aop.) □

Note that Biham and Shamir used the fact $\mathrm{DP}^{+}(\alpha,\beta\mapsto\alpha\oplus\beta)=2^{-\mathsf{w_h}((\alpha\vee\beta)\wedge\mathsf{mask}(n-1))}$ in their differential cryptanalysis of FEAL in [BS91b]. Often this value is significantly smaller than the maximum differential probability $\mathrm{DP}^{+}_{\mathsf{max}}(\alpha,\beta)$. For example, if $\alpha=\beta=2^n-1$, then $\mathrm{DP}^{+}_{\mathsf{max}}(\alpha,\beta)=\frac{1}{2}$, while $\mathrm{DP}^{+}(\alpha,\beta\mapsto\alpha\oplus\beta)=\mathrm{DP}^{+}(\alpha,\beta\mapsto 0)=2^{n-1}$. However, since FEAL only uses 8-bit addition, it is possible to find (α,β)-optimal output differences γ by exhaustive search. This has been done, for example, in [AKM98].

6.3 Log-time Algorithm for Double-Maximum Differential Probability

We next show that the *double-maximum differential probability*

$$\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha):=\max_{\beta,\gamma}\mathrm{DP}^{+}(\alpha,\beta\mapsto\gamma)=\max_{\beta}\mathrm{DP}^{+}_{\mathsf{max}}(\alpha,\beta)$$

of addition can be computed in time $\Theta(\log n)$. (As seen from Algorithm 3, $\mathrm{DP}^{+}_{\mathsf{max}}(\alpha,\beta)$ is a symmetric function and hence $\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha)$ is equal to $\mathrm{DP}^{+}(\alpha,\beta\mapsto\gamma)$ maximized under any two of its three arguments.) In particular, the next theorem shows that $\mathrm{DP}^{+}_{\mathsf{2max}}(\alpha)$ is equal to the (more relevant for the DC) value $\max_{\beta\neq 0}\mathrm{DP}^{+}_{\mathsf{max}}(\alpha,\beta)$ whenever $\alpha\neq 0$. Note that the naive algorithm for the same problem works in time $\Omega(2^{4n})$, which makes it practically infeasible even for $n=16$.

Theorem 5. *For every $\alpha \in \Sigma^n$, Algorithm 5 computes $\mathrm{DP}^+_{\mathsf{2max}}(\alpha)$ in time $\Theta(\log n)$.*

Proof (Sketch). By the same arguments as in the proof of previous theorem, given inputs (α, α), the value $\gamma{:=}\alpha \oplus (C^r(\alpha,\alpha) \wedge \mathsf{mask}(n-1))$ is (α,α)-optimal. We now prove by contradiction that $\mathrm{DP}^+_{\mathsf{2max}}(\alpha) = \mathrm{DP}^+_{\mathsf{max}}(\alpha,\alpha)$. Let $\beta \neq \alpha$ and γ' be such that $\mathrm{DP}^+(\alpha,\beta \mapsto \gamma') > \mathrm{DP}^+_{\mathsf{max}}(\alpha,\alpha)$. By Algorithms 2 and 4, there is an $i < n-1$ such that $\mathsf{eq}(\alpha,\beta,\gamma')_i = 1$ and $C^r(\alpha,\alpha)_i = 1$. But then, on the other hand, since the differential $(\alpha,\beta \mapsto \gamma')$ is possible and $C^r(\alpha,\alpha)_i = 1$, it is also the case that $\mathsf{eq}(\alpha,\beta,\gamma')_{i-1} = 0$. Since $C^r(\alpha,\alpha)_i = 1 \Rightarrow C^r(\alpha,\alpha)_{i-1} = 0$, we have also that $C^r(\alpha,\alpha)_{i-1} = 0$. Therefore $\mathrm{DP}^+(\alpha,\beta \mapsto \gamma') \leq \mathrm{DP}^+_{\mathsf{max}}(\alpha,\alpha)$. □

Straightforwardly, Theorem 5 helps to find many interesting properties of $\mathrm{DP}^+_{\mathsf{2max}}$. For example, $\mathrm{DP}^+_{\mathsf{2max}}(\alpha) = 1$ iff $\alpha \wedge (2^{n-1}-1) = 0$, and $\min_\alpha \mathrm{DP}^+_{\mathsf{2max}}(\alpha) = 2^{-n/2}$. Another consequence is that $\mathrm{DP}^+_{\mathsf{2max}}(\alpha) = \frac{1}{2}$ iff (1) $\alpha \wedge (2^{n-1}-1) = -2^s + 1$ for some $0 \leq s < n$, or (2) $\alpha \wedge (2^{n-1}-1) = 2^s$ for some $0 \leq s < n-1$. For better understanding, all values of $\mathrm{DP}^+_{\mathsf{2max}}(\alpha)$, $n = 8$, are depicted in Fig. 2.

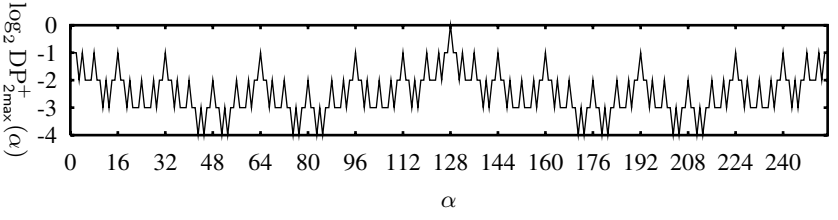


Fig. 2. Tabulation of values $\log_2 \mathrm{DP}^+_{\mathsf{2max}}(\alpha)$, $0 \leq \alpha \leq 255$, for $n = 8$. For example, $\mathrm{DP}^+_{\mathsf{2max}}(64) = \frac{1}{2}$ and $\mathrm{DP}^+_{\mathsf{2max}}(53) = 2^{-4}$

Once again, our results may be compared with the results in [O'C93,O'C95] that show that for an n-bit permutation (resp. composite permutation, controlled by an n-bit string) the expected probability of the maximum nonzero differential is $\leq n/2^{n-1}$ (resp. $\approx 2^{-n}$).

Further Work and Acknowledgments. While we leave practical applications of our results as an open question, we note that our results have already been used in [MY00] for *truncated* differential cryptanalysis of Twofish.

The current work bases somewhat on [Mor00], that had a (correct) linear-time algorithm for DP^+, but with an incorrect proof. We would like to thank Eli Biham for notifying us about the results presented in the full version of [BS91b].

References

[AKM98] Kazumaro Aoki, Kunio Kobayashi, and Shiho Moriai. The Best Differential Characteristic Search of FEAL. *IEICE Trans. Fundamentals*, E81-A(1):98–104, January 1998.

[Ber92] Thomas A. Berson. Differential Cryptanalysis Mod 2^{32} with Applications to MD5. In Ernest F. Brickell, editor, *Advances in Cryptology—CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 71–80. Springer-Verlag, 1993, 16–20 August 1992.

[BS91a] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[BS91b] Eli Biham and Adi Shamir. Differential Cryptanalysis of Feal and N-Hash. In Donald W. Davies, editor, *Advances on Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 1–16, Brighton, UK, April 1991. Springer-Verlag. Full version available from `http://www.cs.technion.ac.il/~biham/`, as of April 2001.

[Dae95] Joan Daemen. *Cipher and Hash Function Design. Strategies based on linear and differential cryptanalysis*. PhD thesis, Katholieke Universiteit Leuven, 1995.

[DGV93] Joan Daemen, René Govaerts, and Joos Vandewalle. Cryptanalysis of 2.5 Rounds of IDEA. Technical Report 1, ESAT-COSIC, 1993.

[Knu99] Lars Knudsen. Some Thoughts on the AES Process. Public Comment to the AES First Round, 15 April 1999. Available from `http://www.ii.uib.no/~larsr/serpent/`, as of April 2001.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov Ciphers and Differential Cryptanalysis. In Donald W. Davies, editor, *Advances on Cryptology — EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38, Brighton, UK, April 1991. Springer-Verlag.

[Miy98] Hiroshi Miyano. Addend Dependency of Differential/Linear Probability of Addition. *IEICE Trans. Fundamentals*, E81-A(1):106–109, January 1998.

[Mor00] Shiho Moriai. Cryptanalysis of Twofish (I). In *The Symposium on Cryptography and Information Security*, Okinawa, Japan, 26–28 January 2000. In Japanese.

[MY00] Shiho Moriai and Yiqun Lisa Yin. Cryptanalysis of Twofish (II). Technical report, IEICE, ISEC2000-38, July 2000.

[NK95] Kaisa Nyberg and Lars Knudsen. Provable Security Against a Differential Attack. *Journal of Cryptology*, 8(1):27–37, 1995.

[O'C93] Luke J. O'Connor. On the Distribution of Characteristics in Composite Permutations. In Douglas R. Stinson, editor, *Advances on Cryptology — CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 403–412, Santa Barbara, USA, August 1993. Springer-Verlag.

[O'C95] Luke O'Connor. On the Distribution of Characteristics in Bijective Mappings. *Journal of Cryptology*, 8(2):67–86, 1995.

Author Index