New Attacks from Old Distinguishers Improved Attacks on Serpent

Marek Broll, Federico Canale, Nicolas David, Antonio Flórez-Gutiérrez, Gregor Leander, María Naya-Plasencia, Yosuke Todo

New Attacks from Old Distinguishers Improved Attacks on Serpent

Marek Broll[1], Federico Canale[1], Nicolas David[2], Antonio Flórez-Gutiérrez[2], Gregor Leander[1], María Naya-Plasencia[2], and Yosuke Todo[3]

[1] Horst Görtz Institute for IT Security, Ruhr University Bochum, Bochum, Germany,
{marek.broll,federico.canale,gregor.leander}@rub.de
[2] Inria, France,
{nicolas.david,antonio.florez-gutierrez,maria.naya_plasencia}@inria.fr
[3] NTT Social Informatics Laboratories, Tokyo, Japan,
yosuke.todo.xt@hco.ntt.co.jp

Abstract. Serpent was originally proposed in 1998 and is one of the most studied block ciphers. In this paper we improve knowledge of its security by providing the current best attack on this cipher, which is a 12-round differential-linear attack with lower data, time and memory complexities than the best previous attacks. Our improvements are based on an improved conditional key guessing technique that exploits the properties of the Sboxes.

Keywords: Differential-linear cryptanalysis · key-recovery · Serpent · conditional differential · conditional linear.

1 Introduction

Symmetric primitives play a vital role in today's secure communication. In a nutshell, their success is certainly based on their outstanding performance on the one hand and the trust in their security on the other hand. The block cipher Serpent [3] is an excellent example of the successful interplay between security and performance considerations. Serpent was a finalist in the Advanced Encryption Standard competition organized by NIST, and – unofficially – ranked second. Its design greatly contributed to our understanding of performance block ciphers by being consequently designed with a bit-sliced implementation in mind. It has also been the target of an impressively large amount of cryptanalysis like [19,5,4,7,8,15,16,14,17,24,23,21] over the years.

All of those attacks are based on variants or generalizations of differential ([10]) and linear ([22]) cryptanalysis, which are two of the fundamental families of attacks on symmetric primitives. Differential cryptanalysis is based on differences between pairs of plaintexts which propagate through the cipher with high probability, while linear cryptanalysis is based on linear approximations which exhibit high correlation. In both cases the aim is to distinguish the cipher from a random permutation by collecting enough plaintext-ciphertext pairs. In [20] a combination of both types of distinguishers was introduced. It is shown that a

Table 1: Summary of working 12-round attacks on Serpent. The adjustment of some of the complexities and attacks with flaws is discussed in the text.

Target	Rounds	Source	Attack type	Complexity		
				Time	Data	Memory
Serpent (256-bit)	12	[24,23]	Multidimensional linear	$2^{253.8}$	$2^{125.8}$	$2^{125.8}$
	12	[24,23]	Multidimensional linear	2^{242}	$2^{125.8}$	2^{236}
	12	[21] [a]	Differential-linear	2^{251}	2^{127}	2^{127}
	12	Section 5.3	Differential-linear	$\mathbf{2^{233.55}}$	$2^{127.92}$	$2^{127.92}$
	12	Section 5.3	Differential-linear	$2^{236.91}$	$2^{125.74}$	$2^{125.74}$
	12	Section 5.3	Differential-linear	$2^{242.93}$	$\mathbf{2^{118.40}}$	$\mathbf{2^{118.40}}$

[a] This attack starts from round 4 instead of round 0.

probability 1 differential and a linear approximation can be combined to obtain a distinguisher that covers more rounds. The technique was extended in [6] to allow for probabilities different from 1 in the differential part.

In this paper we make use of two main observations to construct the best attack on Serpent. We start by extending some ideas presented in [2] for differential-linear attacks on ARX constructions to SPN networks.

In addition, and this is the technically most important part, we use some ideas for improving the key recovery part of these attacks. The idea is to minimize the parts of the key that need to be guessed by closely investigating the involved Sboxes. As we will see, in the case of the coordinate functions of the Serpent Sboxes (and its derivatives) it is possible to deduce information on its output given only partial information on its input. Here, and this allows a significantly more fine-grained approach, we consider not only reductions that are possible for the entire input space, but rather split the input space (repeatedly) into halves by considering linear conditions on the input.

By using these techniques we are able to propose the best known attacks on Serpent, as summarized in Table 1. While the first and third attack yield the best time and data/memory complexity, respectively, the second attack shows that we can get a much better time complexity than the best previously known attacks and better data and memory complexity.

Related Work

Serpent has been the target of multiple reduced-round cryptanalysis efforts ever since it was first introduced. These include linear ([4,16]), multiple and multidimensional linear ([15,14,24]), nonlinear ([23]), boomerang ([19,7]), rectangle ([4,6]), and differential-linear ([8,17,21]) attacks. We will next discuss the best known attacks. All those claim to attack 12 rounds and consider the Serpent version using a 256-bit key.

Flawed Complexity Estimates Before doing so, we would like to point out that unfortunately, quite some previous attacks turned out to be flawed. In some

cases flaws were not describe previously. In all those cases, the overly optimistic complexity estimates were confirmed by the authors of the corresponding papers.

First published 12-round attack. Some differential-linear attacks on 11 and 12-round Serpent were proposed in [17]. However, we have found that the 12-round attack is incorrect (also pointed out in [21] and acknowledged by the authors). Indeed, the attack consists of a 112 keybit guess in the first round, which should allow the attacker to perform the 11-round variant of the attack on the rest of the rounds. This assumption is incorrect, as although the difference in the state after the first round is determined, the input values of the active Sboxes in the second round are unknown, which are required by the 11-round attack as they are part of the plaintext. Any evident workaround would either increase the data complexity beyond the whole codebook or the time complexity over the exhaustive search. The attacks which are presented in this paper are amended versions of this 12-round attack, as they are based on the same distinguisher.

Multidimensional-linear 12-round attacks. A family of multidimensional linear attacks on up to 12 rounds are presented in [24]. The complexity estimates for these attacks were found to be overly optimistic in [23], where new estimates were provided. The original estimates in [24] rely on an overestimated capacity, mistaken time complexity conversion to full encryptions, and very small advantage and success probability. Of the two proposed variants of the 12-round attack, the complexities of the first were corrected to $2^{125.8}$ data complexity, a time complexity of 2^{242} memory accesses, and 2^{108} memory complexity, while the second variant was found to be invalid and no valid corrected version was presented given for it. We have found, however, that the memory complexity for the first attack is also incorrect. The attack consists of 2^{128} repetitions of the 11-round attack, once for each guess of the last round subkey. We can choose between a memory complexity of $2^{125.813}$ to store the data but with a larger time complexity of $2^{253.813}$, or a large memory complexity of 2^{236} with the same time complexity[4]. The values given in Table 1 correspond to the corrected complexities.

Differential-Linear. In a recent work which is independent from ours [21], a new 12-round differential-linear attack on Serpent is proposed that tries to improve the distinguisher bias and the key recovery complexity simultaneously with an algebraic approach. The complexities of their attack can be seen in Table 1, where we have corrected the memory complexity for the sake of comparison: while in the original paper the authors claimed a memory of 2^{99}, they have confirmed to us that they did not take into account the cost of storing the data (as each plaintext-ciphertext pair has to be accessed multiple times), and agree with this comparison. In their paper the authors also indicate the problems with

[4] We contacted the authors with respect to this, but they were unable to provide any further information. We also contacted the authors of [21] and they agreed in a personal communication, which is why these results are not cited in [21].

the attack from [17], and state that they were not able to fix these problems without increasing the complexity beyond that of exhaustive search.

This shows how powerful our new key-recovery techniques are, as they allow to bring the time complexity of this attack below 2^{256}. Even more, we are able to provide the best known 12-round attack on Serpent, as can be seen in Table 1. The time complexity of [21], as can be seen in Table 1 is quite higher than ours even in the version where we have also better data and memory. As a side note our attack is also the only known differential-linear attack on 12 rounds that starts from the first round of Serpent.

Finally, our attack shares some basics with [12]. Indeed, as the authors from [12] point out, our original ideas used specifically for Serpent have inspired that paper's generalized framework used for other ciphers and other attacks than differential-linear ones.

Outline

The paper is structured as follows: Section 2 provides the specifications of Serpent as well as a short introduction to differential-linear attacks. Section 3 provides a first "fixed" version of the 12-round attack which was introduced in [17] whose corrected time complexity is very close to exhaustive search. Section 4 focuses on some useful properties of the Serpent Sboxes, as well as discussing how similar properties should be exploitable in all small Sboxes. Finally, in Section 5 we use these properties in an improved version of the attack which reduces the time complexity by a factor of around 2^{22}.

2 Preliminaries

2.1 Description of the Serpent cipher

Serpent is a block cipher which was introduced in [3] by Anderson, Biham and Knudsen. Its design consists of a substitution-permutation network (SPN) with an internal state of 128 bits. It admits 128, 192 or 256 bit keys. The encryption map consists of a round function which is iterated 32 times. The round function consists of three steps: first the state is XORed with key material, then a layer of of 4-bit Sboxes is applied, and the round ends with a linear transformation. The cipher makes use of the following 8 Sboxes, which were constructed in a pseudo random way to fulfil several cryptographic criteria:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_0(x)$	3	8	F	1	A	6	5	B	E	D	4	2	7	0	9	C
$S_1(x)$	F	C	2	7	9	0	5	A	1	B	E	8	6	D	3	4
$S_2(x)$	8	6	7	9	3	C	A	F	D	1	E	4	0	B	5	2
$S_3(x)$	0	F	B	8	C	9	6	3	D	1	2	4	A	7	5	E
$S_4(x)$	1	F	8	3	C	0	B	6	2	5	4	A	9	E	7	D
$S_5(x)$	F	5	2	B	4	A	9	C	0	3	E	8	D	6	7	1
$S_6(x)$	7	2	C	5	8	4	6	B	E	9	1	F	D	3	A	0
$S_7(x)$	1	D	F	0	E	8	2	B	7	4	C	A	9	3	5	6

We now provide a precise description of the round function, which will be given using the bitsliced notation of [3]. The 128-bit internal state X is represented by four 32-bit words which are denoted X_0, X_1, X_2 and X_3, with $X_j[i]$ being the i-th leftmost bit of word j. The 32 rounds are numbered 0 to 31. Each round consists of the following four steps:

- **Key mixing**. A 128 bit subkey is XORed to the internal state. The subkey at round i will be denoted by $\widehat{K}_i$.
- **Sbox Layer**. Different Sboxes are used depending on the round, in particular $S_{(i \bmod 8)}$ is used at round i. The Sbox Layer operation consists of the application of 32 copies of the Sbox to the internal state. For each $i \in \{0, \ldots, 31\}$ we perform the appropriate Sbox transformation to the 4-bit string $(X_0[i], X_1[i], X_2[i], X_3[i])$. The parallel application of the Sbox in round i will be denoted $\widehat{S}_i$.
- **Linear transformation**. The linear operation is denoted by LT and consists of the following steps:

$$
\begin{aligned}
&X_0 \leftarrow X_0 \lll 13; \quad X_2 \leftarrow X_2 \lll 3\\
&X_1 \leftarrow X_1 \oplus X_0 \oplus X_2; \quad X_3 \leftarrow X_3 \oplus X_2 \oplus (X_0 \ll 3)\\
&X_1 \leftarrow X_1 \lll 1; \quad X_3 \leftarrow X_3 \lll 7\\
&X_0 \leftarrow X_0 \oplus X_1 \oplus X_3; \quad X_2 \leftarrow X_2 \oplus X_3 \oplus (X_1 \ll 7)\\
&X_0 \leftarrow X_0 \lll 5; \quad X_2 \leftarrow X_2 \lll 22
\end{aligned}
$$

Here $\ll j$ denotes a j-bit left shift and $\lll j$ denotes a j-bit left rotation. In round 31 this linear transformation is ommited and an additional subkey $\widehat{K}_{32}$ is XORed to the state instead.

If we denote by $\widehat{B}_i$ the 128-bit value at round i, the cipher operates as follows:

$$
\begin{aligned}
\widehat{B}_0 &\leftarrow IP(P)\\
\widehat{B}_{i+1} &\leftarrow R_i(\widehat{B}_i)\\
C &\leftarrow IP^{-1}(\widehat{B}_{32})
\end{aligned}
$$

where

$$\begin{aligned} IP &\text{ is an initial permutation} \\ R_i(X) &= LT(\widehat{S}_i(X \oplus \widehat{K}_i)) \text{ for } i = 0, \ldots, 30 \\ R_i(X) &= \widehat{S}_i(X \oplus \widehat{K}_i) \oplus \widehat{K}_{32} \text{ for } i = 31 \end{aligned}$$

Key schedule. The key schedule described in [3] turns the 256-bit user key K into 33 128-bit round subkeys, that is, 132 32-bits words of key material. The user key is first written as 8 32-bit words $K = w_{-8}w_{-7}\cdots w_{-1}$. This prekey sequence is extended using the recurrence relation

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-1} \oplus \phi \oplus i) \lll 11,$$

where $\phi =$ `0x9e3779b9`. We then build the sequence k_i from w_i using the Sboxes:

$$\begin{aligned} \{k_0, k_{33}, k_{66}, k_{99}\} &= S_3(w_0, w_{33}, w_{66}, w_{99}) \\ \{k_1, k_{34}, k_{67}, k_{100}\} &= S_2(w_1, w_{34}, w_{67}, w_{100}) \\ &\cdots \\ \{k_{31}, k_{64}, k_{97}, k_{130}\} &= S_4(w_{31}, w_{64}, w_{97}, w_{130}) \\ \{k_{32}, k_{65}, k_{98}, k_{131}\} &= S_3(w_{32}, w_{65}, w_{98}, w_{131}) \end{aligned}$$

We then distribute the 32-bit words k_j to build the 128-bit subkeys K_i:

$$K_i = \{k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3}\}$$

The round key $\widehat{K_i}$ is the result of the application of the fixed initial permutation IP to K_i, hence $\widehat{K_i} = IP(K_i)$.

2.2 Differential-linear Cryptanalysis

We consider a cipher E_K decomposed in two parts F_K and G_K, thus $E_K = G_K \circ F_K$. The ideas developed in [20,6] allow to combine differential properties of F_K and linear properties of G_K to build a more global property on E_K . Let $\delta \longrightarrow \Delta$ be a differential with probability p for F_K. In other words,

$$\text{Prob}_{x,K}\left(F_K(x+\delta) = F_K(x) + \Delta\right) = p.$$

Let $\langle \alpha, y\rangle \oplus \langle \beta, z\rangle$ be a linear approximation of G_K with correlation q:

$$\text{Prob}_{x,K}\left(\langle \alpha, y\rangle \oplus \langle \beta, G_K(y)\rangle = 0\right) - \text{Prob}_{x,K}\left(\langle \alpha, y\rangle \oplus \langle \beta, G_K(y)\rangle = 1\right) = q.$$

In [20,6] it is shown that

$$\text{Corr}_{x,K}\left(\langle \beta, E_K(x)\rangle \oplus \langle \beta, E_K(x \oplus \delta)\rangle\right) = pq^2$$

under certain independence assumptions. This means that E_K can be distinguished from a random permutation with $O\left(p^2q^4\right)$ chosen plaintext-ciphertext pairs. It suffices to generate plaintext pairs $(x, x \oplus \delta)$ and compute the correlation of $\langle \beta, E_K(x) \rangle \oplus \langle \beta, E_K(x \oplus \delta) \rangle$. Note that the presence of truncated differentials and linear hulls will influence the correlation of a differential-linear distinguisher. Furthermore, the independence assumption is not always satisfied, so the transition between the differential and the linear part may have an unexpected correlation. To overcome this issue, several approaches, such as the differential-linear connectivity table (DLCT) [1] have been introduced. One of the most commonly-used techniques is to remove a small number of transition rounds from the differential and the linear paths and estimate their correlation r experimentally, and consider the correlation of the full distinguisher is prq^2. This approach is of course limited to sufficiently large correlations r.

Another improvement we use is the idea introduced in [2] of generating many pairs which verify the differential part of the distinguisher from a single pair. In the paper, it is noted that flipping certain bits in the input will not change the fact that a pair fulfils the differential part of the differential-linear distingusher. In the best case, this reduced the data complexity by a factor of $1/p$.

A differential-linear distinguisher can be used to mount a key-recovery attack by extending it at the top and/or at the bottom with key guesses. We can guess some key material which determines the input difference of the distinguisher from the plaintext and some key material which determines the output parity bit from the ciphertext. By computing the correlation of the differential-linear distinguisher for each guess of the key, we can separate the correct key guess from all the wrong ones. In order to determine the advantage and success probability of our attacks, we will use the models introduced for linear cryptanalysis in [11].

3 First version of the attack

In this section we present a corrected version of the flawed 12-round attack from [17] by considering the correct diffusion on the key recovery rounds along with the idea introduced in [2] of generating multiple good differential pairs.

3.1 Differential-linear distinguisher

Our attack is based on the same differential-linear distinguisher that was used in the flawed 12-round attack in [17]. We start from a "central" distinguisher which starts with a fixed difference at the beginning of round 2, progresses through a three-round truncated differential with probability $p = 2^{-6}$, and then ends with a five-round linear approximation with correlation $q = 2^{-21}$ (we remove the last round of the original distinguisher). The expected correlation for the differential-linear distinguisher is thus $pq^2 = 2^{-48}$. However, experiments with the correlation of the transition rounds between the differential and the linear trail suggest that the actual correlation should be at least $2^{-46.75}$. Our aim for

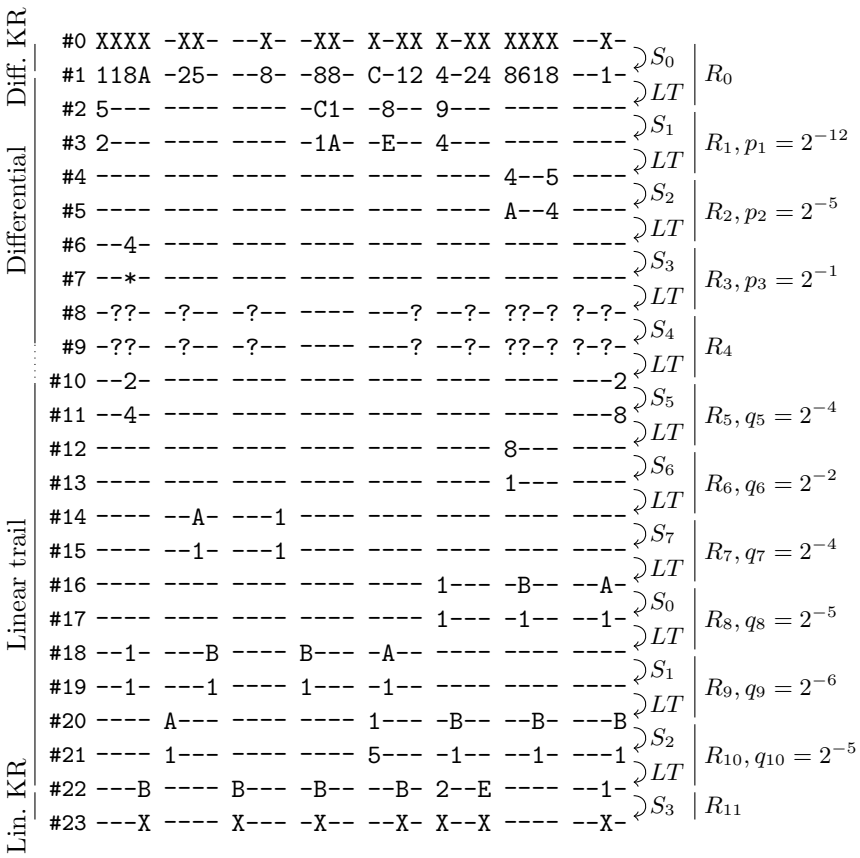


Fig. 1: The basic differential-linear attack on 12-round Serpent obtained by simple extension of the distinguisher, which requires more data than the whole codebook. The hyphen - represents a zero difference/mask in the nibble to improve readability. The nibble difference * is undetermined but is zero in the rightmost bit.

the twelve-round attack is to effectively extend this distinguisher by two rounds at the top and two rounds at the bottom.

If we consider (one of) the best differential transition(s) for each active round 1 Sbox and the best linear approximation for each active Sbox in round 10 (in other words, if we extend the distinguisher to these rounds without any key recovery), the data needed for a reasonable probability of success would be more than $2^{2\cdot(48.75+12+2\cdot 5)} = 2^{141.5}$, which surpasses the size of the codebook. This version of the attack is illustrated in figure 1.

3.2 Improving the first rounds

Our first improvement is inspired by the ideas proposed in [2] in the context of ARX, which we adapt to SPN constructions. The idea is to make sure that some of the differential transitions in round 1 hold for all the data by guessing all the keybits required to know the input values to these Sboxes, instead of only the input differences. For each plaintext, we can deduce a part of the associated plaintexts which will guarantee that the pair always verifies these transitions. This requires guessing more keybits than needed for computing the difference at the beginning of round 1, but allows a reduction in data complexity.

There are five active Sboxes (11,14,17,18 and 31) in round 1, which lead to 20 active Sboxes in round 0. In other words, we have to guess 80 bits of $\widehat{K}_0$ in order to obtain the desired difference at the beginning of round 1. The aim now is to choose some Sboxes in round 1 for which we will also determine the input values, and to do so in a way which optimizes the number of additional keybit guesses. We therefore look at each of the active Sboxes in round 1 and see which additional Sboxes in round 0 would be activated if we decide to guess the input values as well as the differences. We see that in the case of Sboxes 17, 18 and 31, not all Sboxes are active in round 0.

Round 1 Sbox	**Differential Probability**	**Inactive Sboxes in round 0**
11	2^{-2}	$\{3, 16, 19, 22, 23, 27\}$
14	2^{-2}	$\{19, 20, 22\}$
17	2^{-3}	$\{0, 14, 22, 23, 27\}$
18	2^{-2}	$\{2, 10, 19, 23, 24\}$
31	2^{-3}	$\{0, 10, 14, 23\}$

From this table we infer that our efforts should be directed towards Sboxes 11, 17,18 and 31. Sboxes 31 and 17 are particularly interesting as their transition probability is smaller. If we simply fixed the input value of the four Sboxes to one which satisfies the transition with the input difference indicated in the figure, the amount of available data for each key guess would be reduced by a factor of 2^{10}. Instead we consider that the input differences to these Sboxes are variable, as we just need the correct output difference to be satisfied. This also means that the target output difference for round 0 is variable.

Giving this treatment to the four Sboxes would make all Sboxes except for 23 active in round 1. In order to reduce the number of keybit guesses, we will only choose 3 out of these 4 Sboxes, and we will impose some conditions in the input differences in order to obtain the best data complexity: the best choice are Sboxes 31, 18 and 17. Let us point out that at the end of section 5.3 another variant is proposed by considering the four Sboxes that provides the best data complexity. We have chosen two conditions that provide the best trade-off between the reduction of available data and the reduction of keybits to guess.We first impose a 1-bit condition to the difference in state #2 by requiring the difference in bit x_3 in column 17 to be 0. This rejects half of the plaintext pairs, but reduces the number of active bits in the previous round. This condition is

compatible with the original difference, which was 1. Additionally, in column 31 we only consider differences in which the difference in x_0 is zero. This condition is not compatible with the original difference of 5 and rejects 3/4 of the plaintexts, but makes column 29 in the previous round inactive. In the end we keep 2^{-3} of the plaintext pairs, which means we can generate up to 2^{124} differential pairs for each key guess from the whole codebook. This new path is represented in Figure 2.

In addition to the 76 keybits[5] corresponding to the difference, computing the input values to these three Sboxes requires guessing all the other keybits except for Sboxes 23 and 29, which are inactive. This implies a 120 bit key guess in round 0 plus 12 bits in the round 1 for the three active Sboxes in #2 whose input values are required. In the end the amount of pairs required by the attack is reduced by a factor of $2^{2\cdot(3+3+2)-3} = 2^{13}$ with respect to the previous version.

We can summarise the differential pair generation as follows:

- Pick a random plaintext x.
- Guess 120 bits of $\widehat{K}_0$ and 12 bits of $\widehat{K}_1$. With these we can compute the outputs of S_1 in columns 17, 18 and 31, $y[17], y[18]$ and $y[31]$, respectively.
- With these we can compute $\Delta[17] = S_1^{-1}(y[17]) + S_1^{-1}(y[17] + \mathtt{A})$, $\Delta[18] = S_1^{-1}(y[18]) + S_1^{-1}(y[18] + \mathtt{1})$ and $\Delta[31] = S_1^{-1}(y[31]) + S_1^{-1}(y[31] + \mathtt{2})$, which are the input differences to these Sboxes. If $\Delta[17]_3 = 1$ or $\Delta[31]_0 = 1$, we reject the plaintext for this key guess.
- Together with the fixed $\Delta[11] = 9, \Delta[14] = 8$, we have obtained the appropriate input difference for round 1, Δ.
- The associated plaintext is $x' = S_0^{-1}(S_0(x + \widehat{K}_0) + LT^{-1}(\Delta)) + \widehat{K}_0$, and (x, x') is a candidate differential pair.

3.3 Last Rounds

As before, we want to reduce the data complexity by making some changes to round 10. For each of the five active Sboxes (0,5,10,15,27), we determine which bits are required to compute the full output values as opposed to a single parity bit. In other words, for each of these Sboxes we compute which Sboxes would remain inactive in round 11 if we determine the full output value:

Round 10 Sbox	**Correlation**	**Inactive Sboxes in round 11**
0	2^{-1}	$\{2, 4, 6, 9, 10, 12, 14, 15, 16, 17, 19, 20, 21, 24, 26, 27, 29, 30, 31\}$
5	2^{-1}	$\{0, 3, 4, 9, 14, 15, 16, 17, 19, 20, 21, 22, 24, 25, 26, 29, 31\}$
10	2^{-1}	$\{2, 4, 5, 9, 14, 16, 19, 20, 21, 22, 24, 25, 26, 27, 29, 30, 31\}$
15	2^{-1}	$\{0, 2, 3, 4, 6, 7, 9, 10, 14, 19, 21, 24, 25, 26, 27, 29, 30, 31\}$
27	2^{-1}	$\{3, 4, 5, 6, 7, 9, 10, 12, 14, 15, 16, 19, 21, 22, 25, 26, 31\}$

We will use a linear output mask spread between the inputs and the outputs of the S_2 layer in round 10. If we determine the full output (instead of just the

[5] There are 4 less bits to guess as column 29 is now inactive.

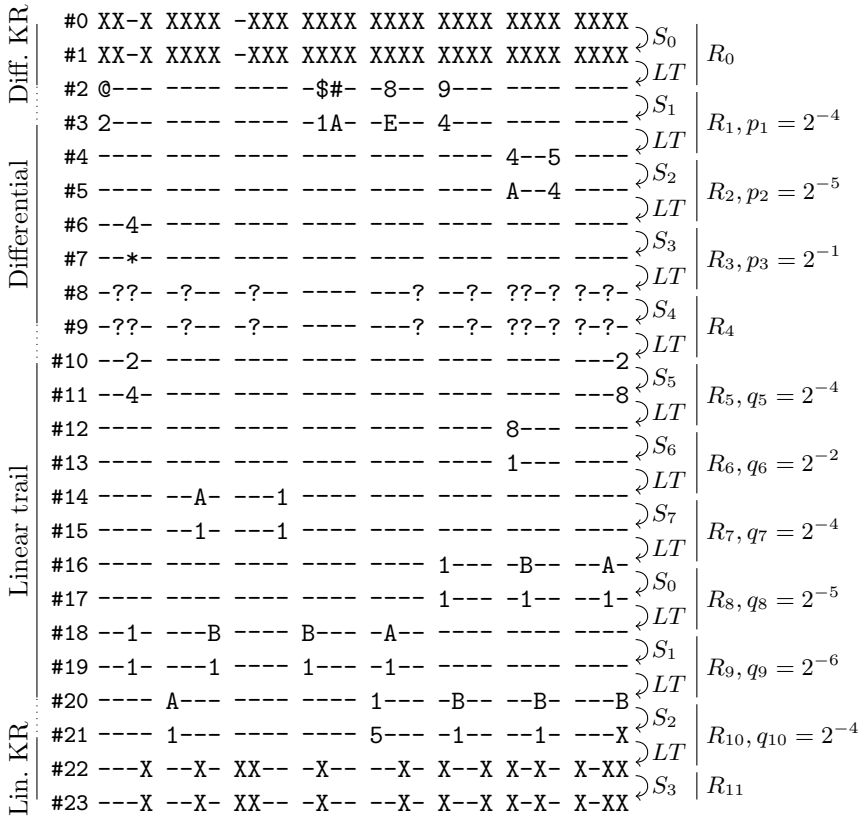


Fig. 2: The slightly improved differential-linear attack on 12-round Serpent with staggered key-recovery. The nibble difference `$` is undetermined but is one in the leftmost bit x_3 because of the differential properties of S_1. The nibble difference `*` is undetermined but is zero in the rightmost bit x_0. The nibble difference `@` is undetermined but is zero in the rightmost bit x_0. The nibble difference `#` is undetermined but is zero in the leftmost bit x_3.

desired parity bit) of Sbox 0, there are 13 active Sboxes in the last round. The other four active Sboxes in round 10 are part of the differential-linear distinguisher. We thus need to guess $4 \cdot 13 = 52$ keybits of $\widehat{K}_{12}$, and 4 bits of $\widehat{K}_{11}$.

The key recovery in the linear part can be performed efficiently by using the FFT algorithm for multiple rounds which was introduced in [18], with a cost of $2 \cdot 52 \cdot 2^{2 \cdot 52}$ additions / substractions and $2^{2 \cdot 52}$ products for each guess of the key in the top and each of the 2^4 key guesses in round 10.

3.4 Complexity analysis

In order to properly evaluate the time complexity, we need to compare the cost of the basic operations of the attack against 12-round Serpent encryptions. We first focus on the distillation phase of the FFT algorithm For each generated plaintext: we perform a two round encryption at the beginning and a one round decryption at the end. However, the partial encryption can be performed for one plaintext x and the result can be reused for all input values of the two inactive Sboxes in round 0. The filtering requires us to process 2^3 plaintexts to find one which satisfies the desired conditions. The cost of processing each plaintext is thus $(2^{-8} \cdot 2^3 \cdot 2/12 + 1/12) = 2^{-3.50}$ encryptions.

For the comparison of a 12-round encryption with one of the arithmetic operations, we note that we can consider $128 \cdot 3 \cdot 12 = 4608$ as a lower bound for the number of bit operations in a 12-round Serpent encryption. On the other hand, a 128 bit addition can be performed with 256 bit operations and a 128 bit product with $128 \cdot \log_2(128) \simeq 961$ bit operations. We thus get a worst-case $2^{-4.17}$ conversion factor for the additions and $2^{-2.26}$ for the products.

Given the distinguisher's correlation of $2^{-48.75-4-2\cdot 4} = 2^{-60.75}$, and using the model from [11], we obtain that with $2^{123.96}$ pairs, ($2^{127.96}$ data before sieving), an advantage of 15 bits is obtained with probability 0.1.

The time complexity of the attack is as follows:

$$\underbrace{2^{120} \cdot 2^{12}}_{\text{Top key guess}} \cdot \Big(2^{123.96} \cdot 2^{-3.5} + \underbrace{2^4 \cdot \left(104 \cdot 2^{-4.17} + 2^{-2.26}\right) \cdot 2^{104}}_{\text{Bottom key guess}}\Big) + 2^{256-15} \simeq 2^{252.46}$$

encryptions. We need $2^{127.96}$ registers for the data, as well as 2^{104} for the distillation tables of the FFT. The overall memory complexity is thus around $2^{127.96}$.

4 Sbox conditions

The purpose of the present Section is to illustrate some conditional properties of the Serpent Sboxes which can improve the differential part of the attack, as well as studying the presence of similar properties in small (4-bit) Sboxes.

4.1 Conditions on the Serpent Sboxes

We consider the first Serpent Sbox, S_0, which is used in round 0. If we denote its input by $x = (x_3, x_2, x_1, x_0)$ and the output by $y = (y_3, y_2, y_1, y_0)$ then the following set of conditional relations always holds:

$$\begin{aligned}
x_2 \oplus x_1 \oplus x_0 = 1 &\Rightarrow y_0 = x_3 \oplus x_2 \oplus x_0 \oplus 1 \\
x_3 \oplus x_2 \oplus x_1 = 0 &\Rightarrow y_1 = x_3 \oplus x_2 \oplus x_0 \oplus 1 \\
x_2 = 1 &\Rightarrow y_2 = x_3 \oplus x_1 \oplus x_0 \\
x_3 = 0 &\Rightarrow y_3 = x_2 \oplus x_1 \oplus x_0 \\
x_3 = 1 &\Rightarrow y_3 = x_3 \oplus x_2 \oplus x_1
\end{aligned}$$

Thanks to these relations, we can compute output bit y_3 without querying the full input. We first query x_3. Depending on this bit, we query either $x_2 \oplus x_1 \oplus x_0$ or $x_3 \oplus x_2 \oplus x_1$ to obtain y_3. This decreases the amount of key material that needs to be guessed in an attack, as in the end we only have to consider four different guesses of the key as opposed to the usual sixteen. More specifically, if $x_i = m_i \oplus k_i$ where (m_0, m_1, m_2, m_3) is the known state before adding the key guess (k_0, k_1, k_2, k_3), then in order to compute the output bit y_3 of S_0, we first guess k_3 and determine the input bit $x_3 = m_3 \oplus k_3$. If $x_3 = 0$, we guess the bit $k_2 \oplus k_1 \oplus k_0$ and, thanks to the above conditions, we have that

$$y_3 = x_2 \oplus x_1 \oplus x_0 = (m_2 \oplus m_1 \oplus m_0) \oplus (k_2 \oplus k_1 \oplus k_0).$$

Otherwise, when $x_3 = 1$ we guess $k_3 \oplus k_2 \oplus k_1$ and determine

$$y_3 = x_3 \oplus x_2 \oplus x_1 = (m_3 \oplus m_2 \oplus m_1) \oplus (k_3 \oplus k_2 \oplus k_1).$$

We can find some similar relations which determine y_0, y_1 and y_2 in the cases which were not covered by the previous set of relations:

$$\begin{aligned}
x_2 \oplus x_1 \oplus x_0 = 0 \text{ and } x_2 = 0 &\Rightarrow y_0 = x_3 \oplus 1 \\
x_2 \oplus x_1 \oplus x_0 = 0 \text{ and } x_2 = 1 &\Rightarrow y_0 = x_0 \oplus 1 \\
x_3 \oplus x_2 \oplus x_1 = 1 \text{ and } x_2 = 0 &\Rightarrow y_1 = x_0 \oplus 1 \\
x_3 \oplus x_2 \oplus x_1 = 1 \text{ and } x_2 = 1 &\Rightarrow y_1 = x_3 \oplus 1 \\
x_2 = 0 \text{ and } x_1 = 0 &\Rightarrow y_2 = x_3 \\
x_2 = 0 \text{ and } x_1 = 1 &\Rightarrow y_2 = x_0 \oplus 1
\end{aligned}$$

With these relations we can determine any one bit of the output of the Sbox with either four (output bit y_3) or six different guesses of key-bits on average. We can construct analogous conditional relations which determine more than one output bit at the same time. A more convenient representation of these conditions can be given with binary trees, as it is explained in the appendix.

The same technique can be applied to reduce the guesses necessary to determine whether a pair $(x, x') = (m \oplus k, m' \oplus k)$ satisfies a certain output difference Δ, i.e., if $S_0(x) \oplus S_0(x') = \Delta$. We consider the function

$$F_\Delta(x) = S_0^{-1}(S_0(x) \oplus \Delta) \oplus x,$$

and we search for conditional relationships which allow us to find a good pair (x, x') by guessing k only partially. The reason is that, if (x, x') is a good pair,

$$F_\Delta(x) = x' \oplus x = m' \oplus m,$$

and knowing $F_\Delta(x)$ and m is therefore enough to recover m'.

We give a partial example for $\Delta = \mathtt{4}$. From the relations of F_4 represented as a tree in Figure 6a, we derive the following (partial) conditions for the case $x_0 = 0$:

$$\begin{aligned}
x_0 = 0 \text{ and } x_1 = 0 &\Rightarrow F_4(x) = \mathtt{C} \\
x_0 = 0 \text{ and } x_1 = 1 &\Rightarrow F_4(x) = \mathtt{5} \oplus x_3\mathtt{2}
\end{aligned}$$

Therefore, in order to guess $F_4(x)$, we have first to guess bit k_0. If $x_0 = m_0 \oplus k_0 = 0$, then we guess k_1 and find that $F_4(x) = m' \oplus m = \mathtt{C}$ if $x_1 = k_1 \oplus m_1 = 1$, i.e. $m' = m \oplus \mathtt{C}$. Otherwise, if $x_1 = k_1 \oplus m_1 = 0$, we guess k_3 and find out that

$$m' = \begin{cases} m \oplus \mathtt{5} & \text{if } x_3 = k_3 \oplus m_3 = 0 \\ m \oplus \mathtt{7} & \text{if } x_3 = k_3 \oplus m_3 = 1 \ . \end{cases}$$

The case for which $x_0 = 0$ can be treated in a similar way. Thanks to this gradual way of guessing k, we obtain the desired information by doing only an average of six guesses instead of sixteen.

4.2 Conditions on general 4-bit Sboxes

We will now show that the presence of linear conditions like the ones we have shown for S_0 is unavoidable if the number of bits of the Sbox is small compared to the number of (linear) restrictions in the input.

Lemma 1. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a permutation (Sbox). If $2^k - k \leq n$, then there exist a subspace U of dimension k of $\mathbb{F}_2^n$ and a vector $u \in \mathbb{F}_2^n$ such that $f|_{U+u}$ has maximum linearity, that is, there exist nonzero linear masks $\alpha \in U + u,\ \beta \in \mathbb{F}_2^n$ for which $\alpha \cdot x + \beta \cdot f(x)$ is constant in $U + u$.*

Proof. Pick any subspace $V \subseteq \mathbb{F}_2^n$ of dimension $n - k$ and set $U = V^\perp$. Let L_V be an $n - k \ \times \ n$ matrix whose row subspace is V and let M_U be a $2^k \ \times \ n$ matrix whose rows are all the elements of U. Note that $x \in U$ if and only if $L_V x = 0$. We denote by $f(M_U)$ the matrix which is generated by applying f to every row of M_U. We can also define $M_U + u$ in the same way for an arbitrary vector $u \in \mathbb{F}_2^n$.

For an arbitrary vector u, $[\alpha, \beta] \in \mathbb{F}_2^{2n}$ is a correlation 1 mask in $U + u$ if

$$\underbrace{\begin{bmatrix} L_V & 0 \\ M_U + u & f(M_U + u) \end{bmatrix}}_{W} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} L_V \alpha \\ c \\ \vdots \\ c \end{bmatrix} \tag{1}$$

for some $c \in \{0, 1\}$. We are only interested in solutions where $\alpha \in U + u$. We set $u = 0$ for convenience, in which case $L_V \alpha$ must be equal to zero.

The matrix W has $2n$ columns and $n - k + 2^k$ rows, so if $n - k + 2^k < 2n$, the system is necessarily under-determined, the kernel of W is nonempty, and all its elements are masks with correlation 1. If $n - k + 2^k = 2n$ and the kernel contains only the trivial mask (that is, if W is a full rank square matrix), then $W^{-1}[0, 1, \ldots, 1]^t$ is a non-trivial mask with correlation 1.

With this lemma we can justify the existence of linear relations in several cases for small Sboxes:

n	1		2			3				4				
k	0	1	0	1	2	0	1	2	3	0	1	2	3	4
$n-2^k+k$	0	0	1	1	0	2	2	1	-2	3	3	2	-1	-8
Relation?	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	?	✗

It is quite clear that in the cases $n = k = 3$ and $n = k = 4$ it is impossible to find such a relation unless the map f is linear, however, in the case $n = 4, k = 3$ we can still prove the existence of these relations:

Lemma 2. *For every permutation $f : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ there exist a subspace U of dimension 3 and a vector $u \in \mathbb{F}_2^4$ such that $f|_{U+u}$ has linearity 8.*

Proof. Without loss of generality we assume that $f(0) = 0$, via affine equivalence. We remove the corresponding row from W and we consider $c = 0$, as for $c = 1$ would get a contradiction. We again set $u = 0$. We pick V at random as above but in such a way that U contains certain vectors a, b, c, d.

W is then a square matrix which either has $\text{rank}(W) < 8$, in which case the kernel of W contains at least one non-trivial solution and we are done, or has rank 8. In the latter case the only solution to the equation is the trivial one. We will now show that it is impossible for the matrix W to have rank 8 if we choose a, b, c, d appropriately.

Since $n = 4$, f cannot be an APN permutation, and it must have a vanishing 2-flat (Theorem 5 in [13]). There exist distinct a, b, c, d for which

$$a + b + c + d = 0 = f(a) + f(b) + f(c) + f(d). \tag{2}$$

This provides a nontrivial relationship between the rows of W which can be used to obtain an equivalent matrix of rank 7 or lower.

4.3 A conditional linear property of S_2

Another improvement we will apply is based on the framework for conditional linear cryptanalysis which was presented in [9]. We consider the Sbox S_2 (which is used in round 10 in the attack) and the linear approximation $\langle B, x\rangle \oplus \langle 1, S_2(x)\rangle$:

	$(y_2, y_0) \neq (1, 1)$												$y_2 = y_0 = 1$			
$\boldsymbol{x}$	0	1	3	4	5	6	9	A	B	C	D	F	2	7	8	E
$\boldsymbol{S_2(x)}$	8	6	9	3	C	A	1	E	4	0	B	2	7	F	D	5
$\boldsymbol{\langle B, x\rangle \oplus \langle 1, S_2(x)\rangle}$	0	1	1	1	1	1	1	0	1	1	1	1	0	1	0	1

If we consider the whole domain, the correlation is $\frac{4-12}{16} = -\frac{1}{2}$. However, under the condition $(y_2, y_0) \neq (1, 1)$ the correlation increases to $\frac{2-10}{12} = -\frac{2}{3}$. We will use this condition to improve the correlation of the linear part of the distinguisher.

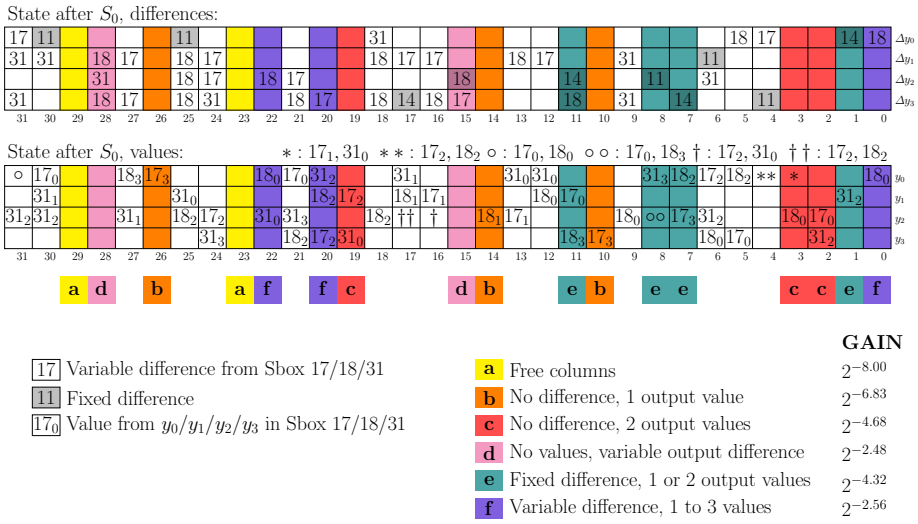


Fig. 3: Reducing the required keybit guesses in the first round.

5 Improved 12-round attack using Sbox properties

In this section, we apply some of the Sbox properties we have discussed in Section 4 to the 12-round attack of Section 3, whose complexity was only barely lower than that of an exhaustive key search. The reduction in time complexity is achieved by a careful analysis of the key recovery in the first two rounds, as well as a small improvement to the linear approximation of round 10 by using conditional linear cryptanalysis ([9]).

5.1 Improving the differential key recovery

We start by looking at the configuration of required differences and values at the output of S_0 in the first round, which is represented in detail in Figure 3. The figure contains both the differences (top half) and the values (bottom half) which are required at the output of the S_0 layer (state #1). On the top half, if a number is written in a cell, then the bit difference on this cell will influence the difference in the column indicated by the number written inside at the input of S_1 in state #2 (more differences might affect it but it is not important and for a better readability of the drawing we just write one here). The color of the background of such a cell indicates whether the difference is fixed (grey) or variable (white). On the bottom half, if an indexed number a_b is written inside a cell, then the bit associated to this cell will influence the value of bit b of the input of Sbox a in the S_1 layer. For space reasons, symbols are used when more than one bit is influenced. We only need the values associated to columns $31, 18$ and 17 as the other two Sboxes are part of the differential and thus only knowledge of the input difference is required. The colors of the columns (also

indexed with lower case letters from a to f) indicate where we can reduce the amount of keybits we guess below four.

The main idea is to exploit the fact that we might not need to guess all the keybits at the input of an Sbox: in some cases the knowledge of a few input bits might be enough to determine all the required information about the output if we guess the key in an orderly manner, as we showed earlier. Furthermore, as the conditional relations introduced in Section 4 describe the desired output bit as a linear combination of the inputs it'll sometimes be possible to XOR the associated keybit guess into a keybit guess for the next round. A more comprehensive explanation and more details in the conditions are provided in A from appendix.

As an example, if we need to determine the input bit of State #2

$$z = y \oplus \widehat{k}_1$$

for some keybit $\widehat{k}_1$ of round 1 and we know that $y = x \oplus \widehat{k}_0$ for some key-bit $\widehat{k}_0$ of round 0 and plaintext bit x (thanks to conditions), then we can combine the two key guesses into one, as $z = x \oplus \widehat{k}_1 \oplus \widehat{k}_0$. This can only be done once for each input bit of S_1 (it is impossible to merge one bit of $\widehat{K}_1$ with several bits of $\widehat{K}_0$ at the same time).

We have considered all the active Sboxes of the state immediately after the application of S_0, as shown in Figure 3. We consider six categories of columns "of interest" depending on the outputs which are required for the attack. For each of this, we are going to compute the multiplicative time factor that we gain with respect to guessing naively, as was done in the previous attack.

Columns of type a (yellow). There are 2 Sboxes (23,29) which are inactive and need no key guessing at all (the same as in the previous attack). These provide gain factor of 2^{-8} with respect to a full subkey guess.

Columns of type b (orange). In columns 10, 14 and 26 we require no output differences, but we need to determine a single output bit from each. Instead of guessing sixteen keybits, we can make use of the properties of S_0:

- Column 26: we only need y_0, which we can determine as follows:
  - If $x_2 \oplus x_1 \oplus x_0 = 0$ then:
    - If $x_1 \oplus x_0 = 0$ then $y_0 = x_3 \oplus 1$.
    - If $x_1 \oplus x_0 = 1$ then $y_0 = x_0 \oplus 1$.
  - If $x_2 \oplus x_1 \oplus x_0 = 1$ then $y_0 = x_3 \oplus x_2 \oplus x_0 \oplus 1$.

  For a given plaintext, we consider six possible guesses of the key instead of sixteen on each column, thus resulting in a 6/16 gain factor for each.
- Column 14: we only need y_2, which can be determined as follows with a gain factor of 6/16:
  - If $x_2 = 0$ then:
    - If $x_1 = 0$ then $y_2 = x_3$.
    - If $x_1 = 1$ then $y_2 = x_0 \oplus 1$.
  - If $x_2 = 1$ then $y_2 = x_3 \oplus x_1 \oplus x_0 \oplus 1$.

- Column 10: we only need y_3 which can be determined as follows with a gain of 4/16:
  - If $x_3 = 0$ then $y_3 = x_2 \oplus x_1 \oplus x_0$.
  - If $x_3 = 1$ then $y_3 = x_3 \oplus x_2 \oplus x_1$.

We obtain a gain factor of $\frac{6}{16} \cdot \frac{6}{16} \cdot \frac{4}{16} \simeq 2^{-4.83}$. In addition, we can absorb the last bit guess for some of the columns into the next round. As bit x_3 of column 17 in S_1 is associated to both Sboxes 10 and 26 in S_0, we can only do this a total of two times (round 1 input bits x_3 in column 17 and x_1 in column 18). The overall gain factor with respect to a full subkey guess is $2^{-4.83} \cdot 2^{-2} = 2^{-6.83}$. The trees associated to these conditions can be seen in Figure 4 from the appendix.

Columns of type c (red). In columns 2, 3 and 19, we require no output difference but two output bits are needed. We proceed as in the previous case, but considering both bits at the same time.

- Column 19: We need y_1 and y_3. Using the trees from Figure 5c in the appendix we can see that determining both bits y_3 and y_1 requires three bits in total. We are also able to absorb one keybit into the next round (x_2 from column 17), but the other bit has to be guessed. The gain for this column is thus $\frac{2 \times 2^3}{2^6} = 2^{-2}$.
- Column 3: We need y_0 and y_2. We can obtain an average guessing cost of 10 as seen in the corresponding tree from Figure 5b, which gives a gain factor of $2^{-0.68}$.
- Column 2: This column (Figure 5a) has the same complexity as 19, as both their trees have the same number of leaves, and also contains two new keybits that might be absorbed: we need 3 input bits to determine the two desired output bits. We can absorb also one keybit in the last linear relation, that can be the keybit from x_0 of column 17 or from x_2 of column 31. We thus also obtain a gain factor of 2^{-2}.

Combining the gains from these Sboxes we obtain a total factor of $2^{-4.68}$.

Columns of type d (pink). Columns 15 and 28 only require no output difference, but this difference is not fixed. There are four possible differences in column 28: `0`, `4`, `A` or `E` (as both Δx_1 and Δx_3 come from difference Δx_2 in column 18 from the next round), which appear with probability 1/4 (this can be deduced from the DDT of S_1 and the output differences for columns 18 and 31).

We first guess the parts of the first round subkey which determine the values at the inputs of columns 18 and 31, so that we know which difference we want at the output of column 28. This is only possible because we only need the output differences from these two columns, and no actual bit values.

We will use the trees built with function F_Δ mentioned in Section 4.1 that are detailed in Figure 6 in the appendix. As an example, if we compute the cost when the desired output difference in column 28 is `4` on average there are 6

possible guesses. The costs for the other non-zero differences are obtained in a similar way and are 2^3 for both. The overall cost becomes:

$$\frac{1}{4}6 + \frac{1}{4}2^3 + \frac{1}{4}2^3 + \frac{1}{4} \cdot 1 \simeq 2^{2.52}$$

instead of 2^4, and implies a gain factor of $2^{-1.48}$.

For column 15 we have two possible output differences: `4` and `C` (as input Δx_3 in column 18 is always 1). For difference `4` the average cost is 6, and for `C` the average cost is 10, which gives a total gain factor of $\frac{6/2+10/2}{16} = 2^{-1}$.

Both gain factors multiply to $2^{-2.48}$.

Columns of type e (turquoise). In columns 1,7, 8 and 11 we have a fixed difference and we also need to determine some output values. We will also use the conditions given by functions F_Δ mentioned in Section 4.1, but now we have also to look at the values of the inputs and outputs to satisfy the conditions. Let us illustrate the example for column 1.

In column 1 we have a fixed output difference of $\Delta = 1$ and we want to determine bit y_1. If we consider function F_1, we can see that there are 8 possible input differences δ: `3`, `6`, `9`, `C`, `B`, `A`, `E` or `F`. In order to determine whether a pair $(x, x \oplus \delta)$ leads indeed to the desired output difference `1`, we always need to guess three key bits: this is because for any fixed δ the set $\{x : S_0(x) \oplus S_0(x \oplus \delta) = \mathtt{1}\}$ is an affine space (of dimension 1) and therefore can be described by three linear conditions, i.e. three key bits are needed in order to verify them. In each of these cases, the three bits needed to guess whether a pair is a good pair are enough to determine y_1. Column 1 has therefore a gain factor of 2^{-1}.

We can do the same thing with columns 7, 8 and 11. In the case of columns 7, some of the possible input differences define an affine space $\{x : S_0(x) \oplus S_0(x \oplus \delta) = \mathtt{8}\}$ of dimension 2, that means only 2 keybits are needed for guessing a good pair with this particular input difference. Taking into account that a further guess is sometimes needed for then determining the desired output values, the number of needed known key bits can be reduced from the full guess and we obtain a gain factor of $2^{-1.13}$. For column 8 we have the output difference that affects one of the needed values, so we will need to guess the four input bits, but we can always absorb the bit x_3 of column 31, so the gain factor is still 2^{-1}. For column 11 all four input bits need to be guessed for each considered input difference, but for two input differences that represent half of the cases (B and 2) two of the guesses independently determine the values of the two needed values. That means that bit y_3, that corresponds to bit x_3, of column 18 can always be absorbed, as it wasn't done before; and that bit y_1, corresponding to bit x_0 of column 17 that was absorbed half of the times in column 2, can now be absorbed the other half of the time. For the other half of the cases only y_3 can be absorbed. The overall gain factor for column 11 is $2^{-1.19}$.

In total these columns generate a gain factor of $2^{-4.32}$.

Columns of type f (purple). We have three columns of this type: variable difference and some values needed. Depending on the value of the difference, we will find some of the cases previously studied.

For half of the cases of column 22, when the difference is zero, we can just use the same tree from Figure 5b, which gives a factor of 10/16, while in the other half of the cases, when the difference is 4, the case is similar as for the e column 8, and we obtain a gain factor of 1/2 absorbing the bit 31_0, which gives $1/2 \cdot \frac{10}{16} + 1/2 \cdot 2^{-1} = 2^{-0.83}$. For column 20, the desired bits are y_0, y_1 and y_3. When the difference is 0, an optimal tree for simultaneously determining these bits is given in Figure 7. From this we obtain a gain factor for this columns of $1/2 \cdot 2^{-0.42} + 1/2 = 2^{-0.19}$.

Column 0 is exactly the case of column 26 when the difference is zero. We can even absorb bit x_0 of column 18 as it hasn't been absorbed before. When the difference is 1, we recover a similar case as for columns of type e, having in this case a gain factor of 1/2. The gain factor of this column is therefore $1/2 \cdot \frac{6}{16} \cdot 1/2 + 1/4 = 2^{-1.54}$.

This gives a total gain factor for these columns of:

$$2^{-0.83} \cdot 2^{-0.19} \cdot 2^{-1.54} = 2^{-2.56}.$$

5.2 Conditional linear cryptanalysis

In the case of the linear key-recovery for the last rounds, even though similar properties exist for S_3 to the ones we have used for S_0, exploiting them is much more complicated than in the differential part, as we are performing the last round key recovery with the FFT. Nevertheless, we can apply conditional properties in the Sbox S_2 of the previous round, round 10, using the framework which was established in [9].

If we look at the linear approximation of Sbox 10, if we impose the condition that its outputs y_0 and y_2 are not both 1 at the same time, then the correlation goes from $-1/2$ to $-2/3$. This would improve the overall correlation by a factor $(2/3 \times (1/2)^{-1})^4 = 2^{1.66}$. As we have to discard the ciphertext pairs which do not verify the condition, we only keep a proportion of $(3/4)^2 = 2^{-0.83}$. With this we can reduce the data complexity to $2^{0.83-1.66}$, resulting in $2^{123.13} \times 2^4 = 2^{127.13}$. However, since the bottleneck of the previous attack was the exhaustive search, we instead keep a similar data complexity of $2^{123.92} \times 2^4 = 2^{127.92}$, so that the advantage increases to 23.

We still have to see how this affects the the key recovery in the final round: with respect to the previous attack, we require output bit y_2 of Sbox 10 in round 10. This additional bit only activates column 6 in the next round. The number of active keybits in the last round is thus 56.

5.3 Complexity and trade-offs

We now compute the time complexity of this improved version of the attack. The data and memory complexities are $2^{127.92}$. If we consider all the gain factors we

have accumulated, the cost of the key recovery part is

$$2^{140-8-6.83-4.68-2.48-4.32-2.56} \cdot \left(2^{123.92-3.5} + 2^4 \cdot 112 \cdot 2^{112} \cdot 2^{-4.17}\right) \simeq 2^{231.91}$$

equivalent encryptions, while the exhaustive search has cost 2^{256-23}. The total time complexity is thus around $2^{233.55}$ 12-round encryptions.

Trade-offs between data and time. We can reduce the data and memory complexities by relaxing one of the conditions in the first rounds, though this increases the time complexity. We remove the condition on the difference in bit x_0 in column 31 of state #2. This reduces the data complexity by a factor 2^2. This changes the differences in columns 29, 26 and 13 from Figure 3: column 29 becomes a column of type e, column 26 becomes a column of type f, and column 13 won't impact the complexity as it was not exploited in the attack.

Column 29. Its previous gain factor was 2^{-4}. For the 3/4 of the cases where the previous bit condition in column 31 no longer holds, we want an ouput difference of `8` (Figure 6e). The new cost for this column is

$$1/4 \times 1 + 3/4 \times 10 = 2^{2.954},$$

which results in a more modest gain factor of $2^{2.954-4} = 2^{-1.04}$.

Column 26. Its previous gain factor was $6/16 = 2^{-1.415}$. However, this will now only apply to 1/4 of the data, which is when the bit condition in column 31 holds. For the other 3/4, the guessing cost is 12/16, results in the cost

$$1/4 \times 6 + 3/4 \times 12 = 2^{3.39},$$

and a new gain factor of $2^{-0.6}$.

Time complexity for a data complexity of $2^{125.74}$*.* By considering the new gain factors and choosing a different amount of data and advantage, we obtain:

$$2^{140-4-1.04-5.40-0.6-4.68-2.48-4.32-2.56} \cdot \left(2^{123.74-3.5} + 2^4 \cdot 112 \cdot 2^{112} \cdot 2^{-4.17}\right)$$

equivalent encryptions for the key recovery and 2^{256-21} encryptions for the exhaustive search, which result in an overall time complexity of $2^{236.31}$. This time the data and memory complexities are $2^{123.74} \times 2^2 = 2^{125.74}$.

Attack with the best data complexity. As we have discussed in Section 3.2, in order to improve the correlation of the distinguisher, we chose to determine three of the five Sboxes active in round 1. We now show how to also determine the fourth one that was considered interesting, allowing to make the data complexity smaller.

In this case there will then be no conditions imposed in the differences of state #2. We can proceed as in the previous attack, and we will obtain one column of type a (23), three columns of type c (2,3 and 19), three columns of type e (0,

7 and 8), and four columns of type f (12, 22, 26 and 29). We compute the gain factor the same way as before, and we obtain for column 29: $2^{-1.607}$, column 26: $2^{-1.192}$, column 23: 2^{-4}, column 22: $2^{-1.35}$, column 19: $2^{1.41}$, column 12: 2^{-1}, column 8: $2^{-1.678}$, column 7: $2^{1.41}$, column 3: $2^{-0.415}$, column 2: $2^{-0.415}$ and column 0: $2^{-1.54}$.

This gives a total gain factor of $2^{-16.023}$.

We will not consider the conditional linear property in the output as otherwise the term related to the FFT would be the bottleneck, and this implies that for a data and memory complexity of $2^{118.40}$, which gives an advantage of 16 bits, we have a time complexity of:

$$2^{128+16-16.023} \cdot \left(2^{118.40-3.5} + 2^4 \cdot 104 \cdot 2^{104} \cdot 2^{-4.17}\right) + 2^{256-16} = 2^{242.93}$$

12-round encryptions.

6 Conclusion

In this paper we improved upon the best known attacks on Serpent. Our improved differential-linear attack has *simultaneously* lower time, data, and memory complexities than the best previous attacks. We would like to stress that our improvements do not modify the underlying distinguisher at all, and only focus on the data selection and, most importantly, the key-guessing procedure. We find it remarkable that by focusing solely on this part, we are able to significantly improve (and thereby actually fix) previous results.

While our approach in this paper is highly specific to Serpent and the differential-linear distinguishers we used, its ideas can be applied more generally, both to other attacks as well as to other ciphers. Indeed, as mentioned above, our work here is the predecessor of the general framework of [12]. Our work also serves as an application of these ideas to differential-linear cryptanalysis, an example of which was not present in [12].

Acknowledgment

This work was partially funded by the DFG, (German Research Foundation) under Germany´s Excellence Strategy - EXC 2092 CASA – 390781972. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 714294 - acronym QUASYModo).

References

1. Bar-On, A., Dunkelman, O., Keller, N., Weizman, A.: DLCT: A new tool for differential-linear cryptanalysis. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23,

2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11476, pp. 313–342. Springer (2019). https://doi.org/10.1007/978-3-030-17653-2_11, https://doi.org/10.1007/978-3-030-17653-2_11

2. Beierle, C., Leander, G., Todo, Y.: Improved differential-linear attacks with applications to ARX ciphers. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12172, pp. 329–358. Springer (2020), https://doi.org/10.1007/978-3-030-56877-1_12
3. Biham, E., Anderson, R.J., Knudsen, L.R.: Serpent: A new block cipher proposal. In: Vaudenay, S. (ed.) Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1372, pp. 222–238. Springer (1998), https://doi.org/10.1007/3-540-69710-1_15
4. Biham, E., Dunkelman, O., Keller, N.: Linear cryptanalysis of reduced round Serpent. In: Matsui, M. (ed.) Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers. Lecture Notes in Computer Science, vol. 2355, pp. 16–27. Springer (2001), https://doi.org/10.1007/3-540-45473-X_2
5. Biham, E., Dunkelman, O., Keller, N.: The rectangle attack - rectangling the Serpent. In: Pfitzmann, B. (ed.) Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding. Lecture Notes in Computer Science, vol. 2045, pp. 340–357. Springer (2001), https://doi.org/10.1007/3-540-44987-6_21
6. Biham, E., Dunkelman, O., Keller, N.: Enhancing differential-linear cryptanalysis. In: Zheng, Y. (ed.) Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2501, pp. 254–266. Springer (2002), https://doi.org/10.1007/3-540-36178-2_16
7. Biham, E., Dunkelman, O., Keller, N.: New results on boomerang and rectangle attacks. In: Daemen, J., Rijmen, V. (eds.) Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2365, pp. 1–16. Springer (2002), https://doi.org/10.1007/3-540-45661-9_1
8. Biham, E., Dunkelman, O., Keller, N.: Differential-linear cryptanalysis of Serpent. In: Johansson, T. (ed.) Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers. Lecture Notes in Computer Science, vol. 2887, pp. 9–21. Springer (2003), https://doi.org/10.1007/978-3-540-39887-5_2
9. Biham, E., Perle, S.: Conditional linear cryptanalysis - cryptanalysis of DES with less than 2^{42} complexity. IACR Trans. Symmetric Cryptol. **2018**(3), 215–264 (2018). https://doi.org/10.13154/tosc.v2018.i3.215-264, https://doi.org/10.13154/tosc.v2018.i3.215-264
10. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. Lecture Notes in Computer Science, vol. 537, pp. 2–21. Springer (1990)

11. Blondeau, C., Nyberg, K.: Improved parameter estimates for correlation and capacity deviates in linear cryptanalysis. IACR Trans. Symmetric Cryptol. **2016**(2), 162–191 (2016), https://doi.org/10.13154/tosc.v2016.i2.162-191
12. Broll, M., Canale, F., Florez-Gutierrez, A., Leander, G., Naya-Plasencia, M.: Generic framework for key-guessing improvements. In: Advances in Cryptology - ASIACRYPT 2021, 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9 2021 (2021)
13. Carlet, C., Charpin, P., Zinoviev, V.A.: Codes, bent functions and permutations suitable for DES-like cryptosystems. Des. Codes Cryptogr. **15**(2), 125–156 (1998). https://doi.org/10.1023/A:1008344232130, https://doi.org/10.1023/A:1008344232130
14. Cho, J.Y., Hermelin, M., Nyberg, K.: A new technique for multidimensional linear cryptanalysis with applications on reduced round Serpent. In: Lee, P.J., Cheon, J.H. (eds.) Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5461, pp. 383–398. Springer (2008), https://doi.org/10.1007/978-3-642-00730-9_24
15. Collard, B., Standaert, F., Quisquater, J.: Improved and multiple linear cryptanalysis of reduced round Serpent. In: Pei, D., Yung, M., Lin, D., Wu, C. (eds.) Information Security and Cryptology, Third SKLOIS Conference, Inscrypt 2007, Xining, China, August 31 - September 5, 2007, Revised Selected Papers. Lecture Notes in Computer Science, vol. 4990, pp. 51–65. Springer (2007), https://doi.org/10.1007/978-3-540-79499-8_6
16. Collard, B., Standaert, F., Quisquater, J.: Improving the time complexity of matsui's linear cryptanalysis. In: Nam, K., Rhee, G. (eds.) Information Security and Cryptology - ICISC 2007, 10th International Conference, Seoul, Korea, November 29-30, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4817, pp. 77–88. Springer (2007), https://doi.org/10.1007/978-3-540-76788-6_7
17. Dunkelman, O., Indesteege, S., Keller, N.: A differential-linear attack on 12-round Serpent. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) Progress in Cryptology - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5365, pp. 308–321. Springer (2008), https://doi.org/10.1007/978-3-540-89754-5_24
18. Flórez-Gutiérrez, A., Naya-Plasencia, M.: Improving key-recovery in linear attacks: Application to 28-round PRESENT. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 221–249. Springer (2020), https://doi.org/10.1007/978-3-030-45721-1_9
19. Kelsey, J., Kohno, T., Schneier, B.: Amplified boomerang attacks against reduced-round MARS and Serpent. In: Schneier, B. (ed.) Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1978, pp. 75–93. Springer (2000), https://doi.org/10.1007/3-540-44706-7_6
20. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Desmedt, Y. (ed.) Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings. Lecture Notes in Computer Science, vol. 839, pp. 17–25. Springer (1994), https://doi.org/10.1007/3-540-48658-5_3

21. Liu, M., Lu, X., Lin, D.: Differential-linear cryptanalysis from an algebraic perspective. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12827, pp. 247–277. Springer (2021), https://doi.org/10.1007/978-3-030-84252-9_9
22. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings. Lecture Notes in Computer Science, vol. 765, pp. 386–397. Springer (1993), https://doi.org/10.1007/3-540-48285-7_33
23. McLaughlin, J., Clark, J.A.: Filtered nonlinear cryptanalysis of reduced-round Serpent, and the wrong-key randomization hypothesis. In: Stam, M. (ed.) Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8308, pp. 120–140. Springer (2013), https://doi.org/10.1007/978-3-642-45239-0_8
24. Nguyen, P.H., Wu, H., Wang, H.: Improving the algorithm 2 in multidimensional linear cryptanalysis. In: Parampalli, U., Hawkes, P. (eds.) Information Security and Privacy - 16th Australasian Conference, ACISP 2011, Melbourne, Australia, July 11-13, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6812, pp. 61–74. Springer (2011), https://doi.org/10.1007/978-3-642-22497-3_5

A Connection to the Tree-Based Key-Recovery of [12] and Conditions used in Section 5

The conditions used in the attack of Section 5 and Section 4.1 can be expressed in terms of a special type of binary decision trees (called affine decision trees in [12] and parity decision trees elsewhere when the bit-length of the output is one). The authors of [12] examine those trees formally.

An affine decision tree is a regular binary tree for which the inner nodes represent the decision rule for evaluation and the leaves represent the function values of all inputs arriving there.

Given a tree for a (non-constant) function f and undetermined (resp. partially determined, key-dependent) input x which has to be evaluated in the key guessing phase, one needs to guess the bit $b = \langle \alpha, x \rangle$ indicated by the root and then follow the appropriate edge to the sub-tree representing the function $f|_{\langle \alpha, x \rangle = b}$ (here: the left sub-tree represents $f|_{\langle \alpha, x \rangle = 0}$ and the right $f|_{\langle \alpha, x \rangle = 1}$). Each leaf represents an affine subspace on which the function is constant and whose codimension is the depth of the corresponding leaf.

Let the tree for f have k leaves with corresponding affine spaces $A_1, \ldots, A_k$ (which we identify with the leaves) and depths $d_1, \ldots, d_k$. When we arrive in a leaf A_i, we have made at most d_i additional guesses during the key guessing phase. The size of each space is $|A_i| = 2^{n-d_i}$. That means that the total cost incurred is

$$\sum_{i=1}^{k} 2^{d_i} 2^{-d_i} = k$$

since 2^{d_i} is the cost of guessing d_i bits and we will end up in A_i for a proportion of 2^{-d_i} inputs. Put differently, the multiplicative gain over guessing all n bits of input is

$$\frac{k}{2^n}.$$

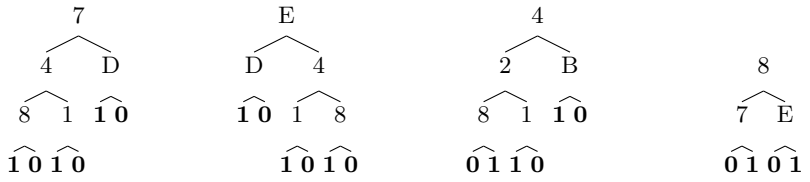
As noted in [12], the goal in most attack scenarios is therefore to minimize the number of leaves (the *size* of the tree).

In the following, we write selection patterns α as hexadecimal numbers, whose binary representation is regarded as an element of $\mathbb{F}_2^4$. The bold numbers refer to leaves, and depending on the number of output bits m, is a hexadecimal number between 0 and $2^m - 1$.

In the following, we give the trees used in the attack of Section 5.

Columns of type b. Let us first look at the set of relations for Sbox S_0 given in Section 4.1. Those relations make up size-minimal trees for coordinate functions of S_0:

$$y_i(x) = \langle 2^i, S(x) \rangle.$$



(a) y_0 (b) y_1 (c) y_2 (d) y_3

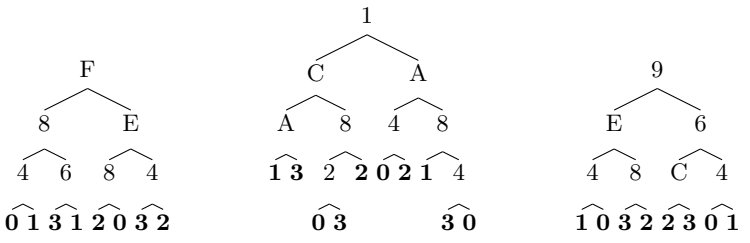
Fig. 4: Trees for $\langle 2^i, S_0(x) \rangle$

In the attack for columns of type b we use the trees from Figures 4a, 4c and 4d.

Columns of type c. For column 19 (type c) one can derive from the conditions given in Section 4.1 that only three bits are necessary to determine both y_1 and y_3 using the tree from Figure 5c. The resulting gain (8/16) is optimal. One additional bit can be absorbed by adding the corresponding keybit from the next round to the final linear relation. The same happens for column 2, giving each of these two columns a gain factor of 2^{-2}. Column 3 has a smaller gain of $2^{-0.68}$.

The trees for columns of type c can be found in Figure 5.

Columns of type d (and e). For columns of type d we have to determine good pairs (x, x') whose output difference $\Delta = S_0(x) + S_0(x')$ belongs to a fixed subset, like for example $\Delta = 4$: Given x, we look at the function F_Δ mentioned in Section 4.1. Its corresponding tree is given in Figure 6a. It has size 6 and therefore a gain factor of 6/16.

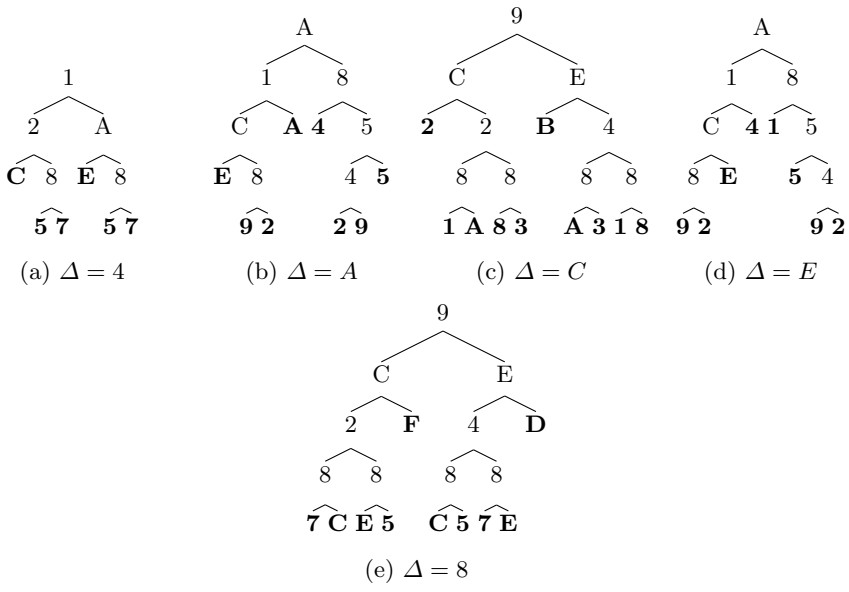


(a) Column 2 (b) Column 3 (c) Column 19

Fig. 5: Trees for columns of type c

When the expected output difference is 0, so must be the input difference. Therefore instead of guessing 4 bits, we need not guess any. Hence we gain a factor of 1/16 in this case. The corresponding tree has one leaf only.

The size-minimal trees for the other three possible differences C (for column 15) and A and E (for column 28) can be found in Figure 6, as well as the tree for the difference 8 of column 29 used for the better memory trade-off.



(a) $\Delta = 4$ (b) $\Delta = A$ (c) $\Delta = C$ (d) $\Delta = E$

(e) $\Delta = 8$

Fig. 6: Trees for $F_\Delta = S_0^{-1}(S_0(x) + \Delta) + x$, columns of type d

A tree for column 20 of type f. For column 20, when the output difference is zero the right input pairs are trivial, while the three output bits must be determined.

The tree shown in Figure 7 shows how to do an optimal guess in order to do that.

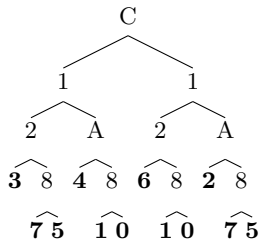


Fig. 7: Tree for column 20, to determine output bits y_0, y_1 and y_3