# Bit-Pattern Based Integral Attack

Muhammad Reza Z'aba[1], Håvard Raddum[2]*, Matt Henricksen[3], and Ed Dawson[1]

[1] Information Security Institute, Queensland University of Technology,
GPO Box 2434, Brisbane, Queensland 4001, Australia
m.zaba@isi.qut.edu.au, e.dawson@qut.edu.au

[2] Selmersenteret , University of Bergen, Norway
haavardr@ii.uib.no

[3] Institute for Infocomm Research, A*STAR,
21 Heng Mui Keng Terrace, Singapore 119613
mhenricksen@i2r.a-star.edu.sg

**Abstract.** Integral attacks are well-known to be effective against byte-based block ciphers. In this document, we outline how to launch integral attacks against bit-based block ciphers. This new type of integral attack traces the propagation of the plaintext structure at bit-level by incorporating bit-pattern based notations. The new notation gives the attacker more details about the properties of a structure of cipher blocks. The main difference from ordinary integral attacks is that we look at the pattern the bits in a specific position in the cipher block has through the structure. The bit-pattern based integral attack is applied to Noekeon, Serpent and PRESENT reduced up to 5, 6 and 7 rounds, respectively. This includes the first attacks on Noekeon and PRESENT using integral cryptanalysis. All attacks manage to recover the full subkey of the final round.

**Key words:** Block ciphers, integral cryptanalysis, Serpent, Noekeon, PRESENT.

## 1 Introduction

The integral attack [11] is the basis for the best attacks on the AES, and has become standard in a cryptanalyst's toolbox. The basic idea of the attack is to analyze how a specified property of a set of plaintexts will evolve through the encryption algorithm and to use the existence of that property to verify key guesses. Up until now, integral attacks have not been thought suitable for bit-based ciphers. In these attacks the plaintext bytes are chosen to be constant, or take on all values through the set of texts. The reason for this choice is that it is unaffected by the application of a bijective S-box substituting bytes.

Using the traditional approach on a bit-oriented cipher, the bits output from an S-box are not treated as a block. This normally implies that any all-values

[*] This work done while visiting the Information Security Institute, Queensland University of Technology, Australia.

property of the S-box output will be subsequently destroyed by the linear layer. In order to address this issue, we introduce a new bit-pattern based notation. Each bit position within a structure holds a specific sequence of bit '0' and '1'. The pattern in which the bit sequence is repeated serves as the basis of the notation. This means that the *order* of the texts in a bit-pattern based integral attack plays an important part, in contrast to the usual integral attack where the texts are regarded as an unordered set. This allows an attacker to gain knowledge of bit patterns in the set of texts through some encryption rounds. Instead of inputting all possible values into a single S-box in the first round, the bit-pattern based structure is constructed such that the active bits are spread over more than one S-box.

The bit-pattern based integral attack manage to penetrate up to 5 (out of 16), 6 (out of 32) and 7 (out of 31) rounds of Noekeon [9], Serpent [1] and PRESENT [7], respectively. To the best of our knowledge, this is the first integral cryptanalysis on Noekeon and PRESENT reported in the literature. For all three ciphers, detailed analysis by the designers show the minimal complexity for a successful differential attack on a few rounds. Bit-pattern based integral cryptanalysis gives much better results for these reduced-round ciphers. Note, however, that differential cryptanalysis can be easily extended to more rounds whereas integral cryptanalysis can not be extended beyond a certain point.

This document is organized as follows. The integral attack and the new notations are explained in Section 2. The attacks on Noekeon, Serpent and PRESENT are presented in Section 3. Section 4 highlights some related work. Discussions and conclusion are given in Section 5.

## 2 Bit-Pattern Based Integral Attack

An integral attack works by choosing a set of plaintexts, where some bit positions take on all values through the set, and the other bits are chosen to be arbitrary constants. The set of cipher blocks used in an integral attack is commonly referred to as a *structure*. The part of the structure that takes on all values usually forms the input to one or more bijective S-boxes in the first round. Then two properties are achieved by the structure: it is unaffected by key-addition, and it is unaffected by the application of bijective S-boxes. The diffusion of the cipher will however mix the bits, so after some time the inputs of some S-boxes will not have the constant or all-values property. When the cipher is byte-oriented (like the AES), the bits in the output of an S-box are treated as a block, and only mixed with blocks of bits that have either the all-values or the constant property. This might delay the destruction of these properties in the structure, and lead to good attacks.

For a bit-based cipher, the bits in the output of one S-box are treated independently and are not mixed in a way that respects the S-box boundaries. Hence the input bits to an S-box in the next round will have some constant bits, and some bits that are not constant. The useful properties of a structure will then be lost already in the second round. We overcome this and make integral attacks possible on bit-based cipher by introducing a more refined notation for the bits in a structure.

### 2.1 Pattern-Based Notations

In our bit-pattern based approach, the status of each single bit position within the overall structure is treated independently. Each bit position in a plaintext structure holds a specific sequence of bit '0' and/or '1'. The pattern in which the bit sequence is repeated forms the foundation of the bit-based notations. The following describes the notation.

- The pattern $\mathbf{c}$ in a position means that all bits in this position within the structure consists of only bit '0' or '1'. This pattern is called a **constant** bit pattern.
- The pattern $\mathbf{a}_i$ in a position means that the first block of $2^i$ consecutive bits in this position are constant, and the next block of $2^i$ consecutive bits all have the opposite value of the first. The alternating values of bits in $2^i$-blocks is repeated throughout the structure. This pattern is called an **active** bit pattern.
- The pattern $\mathbf{b}_i$ in a position means that blocks of $2^i$ consecutive bits in this position are constant, but the values of the blocks are not necessarily repeated in an alternating manner.
- The pattern $\mathbf{d}_i$ in a position means that the bits in this position may hold either a $\mathsf{c}$ (constant) or an $\mathsf{a}_i$ (active) pattern. This pattern is called a **dual** bit pattern.

If the XOR sum of all the bits in one pattern equals 0, we say that the pattern is *balanced*. Furthermore, if the cipher block which is the XOR sum of all the texts in a structure only has 0-bits we say that the structure is balanced. All patterns described above are balanced, except for the $\mathsf{b}_0$-pattern which may or may not be balanced. In fact, any bit-string fulfills the definition of a $\mathsf{b}_0$-pattern. To make a distinction between balanced and unbalanced $\mathsf{b}_0$-patterns, we will write $\mathsf{b}_0^*$ when we know that the pattern is balanced and $\mathsf{b}_0$ otherwise.

As an example, possible values of a 4-bit text structure with the patterns $\mathsf{a}_0\mathsf{a}_3\mathsf{c}\mathsf{a}_2$ are $\{6_x, E_x, 6_x, E_x, 7_x, F_x, 7_x, F_x, 2_x, A_x, 2_x, A_x, 3_x, B_x, 3_x, B_x\}$. Table 4 in the Appendix lists out the possible values of $\mathsf{c}$, $\mathsf{a}_i$ and some $\mathsf{b}_i$ patterns in a structure of $2^4$ texts.

### 2.2 Tracing Bit Patterns Through the Cipher

Bit-patterns will be XORed together in the linear operations of the cipher. The following properties are easy to verify.

- $\mathsf{c} \oplus \mathsf{p} = \mathsf{p}$ for any pattern $\mathsf{p}$.
- $\mathsf{a}_i \oplus \mathsf{a}_i = \mathsf{c}$.
- $\mathsf{p}_j \oplus \mathsf{q}_i = \mathsf{b}_i$ for $j > i$ and $\mathsf{p}, \mathsf{q} \in \{\mathsf{a}, \mathsf{b}\}$. If $i = 0$ the right-hand side will be $\mathsf{b}_0^*$.

- $\mathsf{p} \oplus \mathsf{b}_0^* = \mathsf{b}_0^*$ when $\mathsf{p} \neq \mathsf{b}_0$.

These rules of XOR addition will be used when analyzing how the bit-patterns in the cipher block evolves through the linear parts of a cipher.

When the bit-patterns pass through an S-box, every output-bit of the S-box will have a $\mathsf{b}_i$-pattern where $i$ is the smallest index found in the input patterns. This is because blocks of $2^i$ inputs will all have the same value, and so the output values will also appear in blocks of $2^i$ equal values.

That is all that can be said in general when patterns pass through an S-box, but there is another fact that can be useful when analyzing the effect an S-box has on input patterns. It is summed up in the following lemma.

**Lemma 1.** *Consider $m$ bit sequences, expressed as linear combinations of $\mathsf{a}_i$-patterns $l_1, \ldots, l_m$, where $i \leq n$. Write this using matrix notation as $M\mathsf{a} = \mathbf{l}$, where $\mathsf{a} = (\mathsf{a}_0, \ldots, \mathsf{a}_n)^T$ and $\mathbf{l} = (l_1, \ldots, l_m)^T$. The different values for the $m$ bits found in the same position in the sequences lie in an affine space of size $2^{rank(M)}$.*

*Proof:* Let $r = m - \text{rank}(M)$. Then there exists $r$ linearly independent vectors $\mathbf{v}_1, \ldots, \mathbf{v}_r$ such that $v_i M = \mathbf{0}$, $i = 1, \ldots, r$. Since $\mathsf{a}_i \oplus \mathsf{a}_i = \mathsf{c}$ in our context, a 0-row in $M$ corresponds to the constant pattern. This means that all possible values of the $m$ bits lie in an affine space cut out by the $r$ linear equations given by $\mathbf{v}_1, \ldots, \mathbf{v}_r$ and $r$ right-hand sides. The size of this space is $2^{m-r}$ and the lemma follows. □

The lemma above can be helpful when determining whether the balancedness of a structure is lost through the application of an S-box. Assume we have an $m$-bit S-box and a structure of $2^n$ texts where $m > n$, and assume the input bits to the S-box are expressed as linear combinations of $\mathsf{a}_i$-patterns. Suppose Lemma 1 tells us the inputs to the S-box lie in an affine space of dimension smaller than $n$. Then each distinct input value will occur an even number of times, and so each distinct output value will occur an even number of times. Hence the balancedness will not be lost after the S-box.

### 2.3 Generic Bit-Pattern Based Integral Attack

Here we describe a generic bit-pattern based integral attack that can be used on Noekeon, Serpent and PRESENT. These ciphers are similar in structure, so we will use the same notation on all of them.

The input to round $i$ is denoted by $X_i = (x_0^i, x_1^i, x_2^i, x_3^i)$ where $X_0$ is the plaintext. The input and output of the S-box layer in round $i$ are denoted by $Y_i = (y_0^i, y_1^i, y_2^i, y_3^i)$ and $Z_i = (z_0^i, z_1^i, z_2^i, z_3^i)$, respectively. The round $i$ subkey is denoted by $K_i = (k_0^i, k_1^i, k_2^i, k_3^i)$. The blocks $X_i, Y_i, Z_i$ and $K_i$ consist of four 32-bit words for Noekeon and Serpent, and four 16-bit words for PRESENT. In every word, the rightmost bit is Bit '0' and $x_{[\ell]}$ denotes the $\ell$-th bit of $x$. All non-linear components in these ciphers are composed of $4 \times 4$ bijective S-boxes.

The attacker first finds a structure of plaintexts, and sees how the bit-patterns of the structure become affected through the cipher. Just before the S-box layer in some round the structure will be balanced, but the balancedness is expected to be destroyed after the S-box layer. If this happens in round $r$, the following equation must hold:

$$\bigoplus_{j=0}^{m-1} Y_r^{(j)} = \bigoplus_{j=0}^{m-1} S^{-1}(Z_r^{(j)}) = 0 \qquad (1)$$

where $m$ is the size of the structure. We then guess enough key material so we can partially decrypt the ciphertexts to find all the bits coming out of one of the S-boxes in round $r$, and use Equation (1) to verify the guess.

Equation (1) puts a 4-bit condition on the guess, so we expect the number of possible key-bit guesses to be reduced by a factor $2^{-4}$. If we are guessing on $k$ key-bits at the same time, we will then need approximately $\lceil k/4 \rceil$ structures to identify the correct parts of the round keys used in the last rounds.

This can be summed up in Algorithm (1), where we assume we need to guess $k$ bits from the last round key(s).

```
Precomputation
Analyze round function to identify distinguisher;
begin
    Choose a structure of plaintexts that matches distinguisher;
    Encrypt all plaintexts in structure and get corresponding ciphertexts;
    Initialize an array $A[]$ of size $2^k$ bits with all ‘1’s;
    Set $v = 0$;
    while number of entries such that $A[v] = 1$ is greater than one do
        Partially decrypt all ciphertexts using the value $v$ as partial subkey bits
        to find the output bits of one S-box in round $r$;
        if Equation (1) does not hold then
            set $A[v] = 0$;
        end
        $v = v + 1$;
    end
    Output value $v$ for which $A[v] = 1$ as correct subkey bits;
end
```

**Algorithm 1**: Algorithm for basic attack

We may also extend an attack by one round by adding one round in the beginning. This can be done by letting the bits in the structure have a specific pattern at the input of the second round, instead of in the plaintexts. These patterns are then traced backwards through the first round, until they meet the output of the S-box layer in the first round. S-boxes that have a sum of active patterns in its output bits are called active S-boxes. By specifying a value of the starting bit for patterns in the output of the active S-boxes, we specify some values of these outputs, and can find the values of the inputs. Next we guess the value of the bits in the key used for pre-whitening that affect the active S-boxes in the structure. This allows us to find the structure of plaintexts that will have the specific bit-pattern at the input of the second round, when the guess of bits in the pre-whitening key is right.

If we need to guess $k$ bits from the pre-whitening key, we must expect to use $2^k$ structures before we get one with the specified patterns in the second round. This increases the number of chosen plaintexts needed, but it may involve a smaller guess on key-material than would be needed by adding a round at the end.

## 3 Application on Noekeon, Serpent and PRESENT

We have used bit-pattern based integral cryptanalysis on the block ciphers Noekeon, Serpent and PRESENT. Here we show how the attacks worked.

### 3.1 Noekeon

Noekeon [9] accepts a 128-bit block of plaintext $X_0$ and a 128-bit key. The 128-bit block of ciphertext $X_{17}$ is produced after iterating a round function 16 times, followed by a final output function. The round function consists of two linear layers, $L_0$ and $L_1$, and one non-linear layer $S$. The final round involves only $L_0$. The encryption scheme of Noekeon can be depicted as:

$$\begin{aligned} X_{i+1} &= L_1^{-1}(S(L_1(L_0(X_i, K)))), i = 0, 1, \ldots 15 \\ X_{17} &= L_0(X_{16}, K) \end{aligned}$$

Figure 1 illustrates the round function of Noekeon. The same round subkey is used in every round. Let $x \lll i$ and $x \ggg i$ imply the rotation of the word $x$ by $i$ bits to the left and right, respectively. The linear layer $L_0$ of Noekeon is described as:

$$t_0^i = (x_0^i \oplus c^i \oplus k_0 \oplus u^i) \tag{2}$$

$$t_1^i = (x_1^i \oplus k_1 \oplus v^i) \tag{3}$$

$$t_2^i = (x_2^i \oplus k_2 \oplus u^i) \tag{4}$$

$$t_3^i = (x_3^i \oplus k_3 \oplus v^i) \tag{5}$$

where $u^i = R(p^i)$, $v^i = R(q^i)$, $p^i = x_1^i \oplus k_1 \oplus x_3^i \oplus k_3$, $q^i = x_0^i \oplus c^i \oplus x_2^i$, $R(x) = x \oplus (x \lll 8) \oplus (x \ggg 8)$ and $c^i$ is a round constant. $L_1$ simply consist of three rotations of the words in the cipher block $(y_0^i, y_1^i, y_2^i, y_3^i) = (t_0^i, t_1^i \lll 1, t_2^i \lll 5, t_3^i \lll 2)$.

**3.5-round Distinguisher** Prepare a structure of $2^{16}$ plaintexts:

$$X_0^{(j)} = (x_0^{0(j)}, x_1^{0(j)}, x_2^{0(j)}, x_3^{0(j)}) = (j \| c_0, R(j \| c_1), c_2, R(j \| c_3))$$

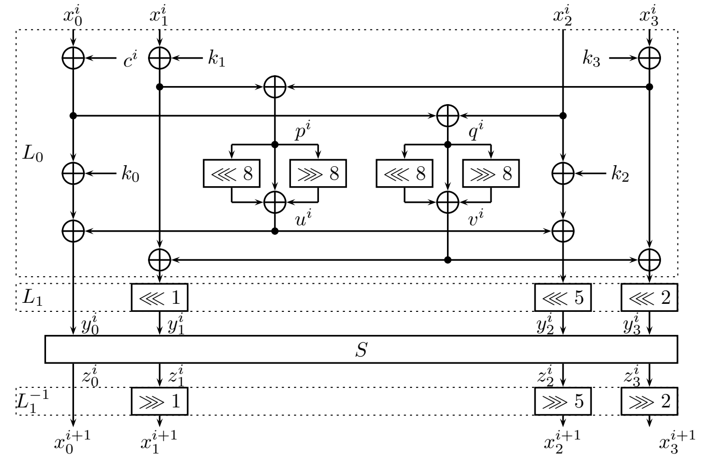

**Fig. 1.** Round function of Noekeon in Round $i$

where $0 \leq j \leq 2^{16} - 1$, $c_0, c_1, c_3$ are arbitrary 16-bit constants and $c_2$ is a 32-bit constant. By consulting Equations (2),(3),(4) and (5), it can be observed that $u^i$ will become a constant and $v^i$ will cancel the active bits in $x_1^0$ and $x_3^0$. This leaves the 16 leftmost bits of $y_0^i$ to hold active patterns, i.e. $\mathsf{a}_{15}\mathsf{a}_{14}\ldots\mathsf{a}_0$. All other bits hold $\mathsf{c}$ patterns. The propagation of bit patterns in this distinguisher is shown in Figure 4 in the appendix.

There are 16 active S-boxes at the input of $S$ in the first round. The remaining 16 S-boxes receive an all $\mathsf{c}$ input patterns. Each active S-box has two inputs which differ only in the leftmost bit. There exists a partial differential through the S-box $8_x \rightarrow w \| 3_x$ with probability 1, where $w \in \{0_x, 1_x, 2_x, 3_x\}$. As a consequence, the 16 leftmost bits of both $z_2^{0(j)}$ and $z_3^{0(j)}$ assume the same $\mathsf{a}_i$ pattern as the leftmost bit of the input. The rest of the output bits of the active S-boxes hold a $\mathsf{d}_i$ pattern where $0 \leq i \leq 15$.

In the second round, the linear combinations of $\mathsf{c}$ and $\mathsf{a}_i$ bits inside $L_0$ guarantee that no $\mathsf{c}$ pattern remains in any bit position. Note that the partial differential plays a critical role to ensure that this property occurs with certainty. Every bit of $u^1$ and $v^1$ contains at least one $\mathsf{a}_i$ pattern. $L_1$ ensures that all bit patterns in every column are linearly independent. According to Lemma 1 there are therefore 16 distinct values in the input and output of every S-box, which are repeated $2^{12}$ times. After the linear operations in the second round, the number of times each distinct value appears in the input of any S-box is still even, so the bits assume a $\mathsf{b}_0^*$ pattern after the S-box layer.

Experimentally, it has been verified that $L_0$ and $L_1$ in the third round do not cause any value in any input to an S-box to occur an odd number of times. At the input of $S$, the number of different inputs into each S-box is even and therefore, the number of different outputs is also even. This causes the structure to remain balanced after $S$.

In the fourth round, the balancedness of the structure is ensured through $L_0$ and $L_1$, but is expected to be destroyed after the application of $S$.

**Key Recovery** The 3.5-round distinguisher can be used to attack four and five rounds of Noekeon using the attack strategy described in Section 2.3.

The key recovery procedure in a 4-round attack is a straightforward process. Once the distinguisher is available, Equation (1) must hold for $m = 2^{16}$ and $r = 3$.

The following equations provide the output bits of the S-boxes in the fourth round:

$$z^{3(j)}_{0[\ell]} = (x^{4(j)}_0 \oplus R(x^{4(j)}_1 \oplus x^{4(j)}_3) \oplus c^4)_{[\ell]} \oplus k_{0[\ell]} \tag{6}$$

$$z^{3(j)}_{1[\ell]} = (x^{4(j)}_1 \oplus R(x^{4(j)}_0 \oplus x^{4(j)}_2))_{[\ell-1]} \oplus A_{1[\ell-1]} \tag{7}$$

$$z^{3(j)}_{2[\ell]} = (x^{4(j)}_2 \oplus R(x^{4(j)}_1 \oplus x^{4(j)}_3))_{[\ell-5]} \oplus k_{2[\ell-5]} \tag{8}$$

$$z^{3(j)}_{3[\ell]} = (x^{4(j)}_3 \oplus R(x^{4(j)}_0 \oplus x^{4(j)}_2))_{[\ell-2]} \oplus A_{3[\ell-2]} \tag{9}$$

where $[\ell + n]$ is computed modulo 32 and $A_{1[\ell]}$ and $A_{3[\ell]}$ are linear combinations of seven key bits as follows:

$$\begin{aligned} A_{i[\ell]} &= (R(k_0 \oplus k_2) \oplus k_i)_{[\ell]} \\ &= (k_0 \oplus k_2)_{[\ell]} \oplus (k_0 \oplus k_2)_{[\ell+8]} \oplus (k_0 \oplus k_2)_{[\ell-8]} \oplus k_{i[\ell]}. \end{aligned} \tag{10}$$

For the 4-round attack, we need to guess on 4 bits of key material at the same time; the bits $k_{0[\ell]}$ and $k_{2[\ell-5]}$ and the values of the linear combinations $A_{1[\ell-1]}$ and $A_{3[\ell-2]}$. This means we should need approximately one structure to identify a correct guess, in practice we sometimes need 2. This needs to be repeated 32 times to get 128 bits of key material from the last round key. After the correct values for $k_{0[\ell]}$, $k_{2[\ell]}$, $A_{1[\ell]}$ and $A_{3[\ell]}$ are identified for $\ell = 0, 1, \ldots, 31$, Equation (10) is rearranged and solved to uncover the unknown bits in $k_1$ and $k_3$. The attack requirements are $2 \times 2^{16} = 2^{17}$ chosen plaintexts and $2 \times 2^{16} \times 2^4 \times 32 = 2^{26}$ partial decryptions.

In a 5-round attack, the values of the outputs from one S-box in the third round can be obtained by guessing 92 selected bits of information from the keys used in round 5 and 4. Here we make use of the fact that Noekeon uses the same key in every round, so the key material we need to guess in round 4 overlaps with what we need to guess in round 5.

In order to correctly identify all the 92 bits, we have to use 23 different structures. The remaining $128 - 92 = 36$ bits can be found by exhaustive search. The number of plaintexts required is therefore $23 \times 2^{16} \approx 2^{20.6}$ chosen plaintexts. The time complexity for the attack is $(2^{92} + 2^{88} + \ldots + 2^4 + 1) \times 2^{16} + 2^{36} \approx 2^{108.1}$ partial decryptions. Memory is required for storing $2^{92}$ bits indicating possible guesses remaining, thus the memory requirement is $2^{89}$ bytes.

### 3.2 Serpent

Serpent [1] is a 128-bit block cipher with key sizes between 0 to 256 bits. It has 32 rounds and can be represented in non-bit-sliced and bit-sliced version. We focus on the bit-sliced version of Serpent. The round function is composed of a key mixing layer, a non-linear S-box layer $S_i$, and a linear transformation layer $L$. In the last round, the linear transformation is replaced with a key mixing layer. The round function of Serpent in Round $i$ is depicted in Figure 2. The cipher can be expressed by the following equations:

$$\begin{aligned} X_{i+1} &= L(S_{i\ mod\ 8}(X_i \oplus K_i)), i = 0, 1, \ldots 30 \\ X_{32} &= S_7(X_{31} \oplus K_{31}) \oplus K_{32}. \end{aligned}$$

Serpent has eight different $4 \times 4$ S-boxes. Round $i$ uses S-box $S_{i\ mod\ 8}$ 32 times in parallel. The input is taken one bit from each 32-bit word of the same bit position. The linear transformation of Serpent can be expressed as:

$$\begin{aligned} x^{i+1}_{0[\ell]} &= z^i_{0[\ell-18]} \oplus z^i_{1[\ell-6]} \oplus z^i_{0[\ell-19]} \oplus z^i_{2[\ell-9]} \oplus \\ &\quad z^i_{3[\ell-12]} \oplus z^i_{2[\ell-15]} \oplus (z^i_{0[((\ell-12)\ll 3)-13]}) \\ x^{i+1}_{1[\ell]} &= z^i_{1[\ell-1]} \oplus z^i_{0[\ell-14]} \oplus z^i_{2[\ell-4]} \\ x^{i+1}_{2[\ell]} &= z^i_{2[\ell-25]} \oplus z^i_{3[\ell-29]} \oplus z^i_{2[\ell]} \oplus (z^i_{0[((\ell-29)\ll 3)-13]}) \oplus \\ &\quad z^i_{1[((\ell-22)\ll 7)-1]} \oplus z^i_{0[((\ell-22)\ll 7)-14]} \oplus z^i_{2[((\ell-22)\ll 7)-4]} \\ x^{i+1}_{3[\ell]} &= z^i_{3[\ell-7]} \oplus z^i_{2[\ell-10]} \oplus z^i_{0[((\ell-7)\ll 3)-13]} \end{aligned}$$

where $[\ell \ll n] = [\ell - n]$ is not computed modulo 32. If $(\ell \ll n) < 0$ then the bit at the position is a zero bit.

The cipher has endured extensive cryptanalysis [12,10,2,3,4,5,8] but no analysis on integral attack has been reported.

**3.5-round Distinguisher** Serpent reduced to 3.5 rounds can be distinguished from a random permutation by choosing a structure of $2^{10}$ plaintexts. The plaintexts are chosen such that the ten most significant bits of $x^0_2$ hold all possible 10-bit values. The rest of the plaintext bits hold constant values such as the following:

$$X^{(j)}_0 = (x^{0(j)}_0, x^{0(j)}_1, x^{0(j)}_2, x^{0(j)}_3) = (c_0, c_1, j \| c_2, c_3)$$

where $0 \leq j \leq 2^{10}-1$, $c_0, c_1, c_3$ are 32-bit constants and $c_2$ is a 22-bit arbitrary constant. The ten leftmost bits of $x^{0(j)}_2$ therefore hold the pattern $\mathsf{a}_9\mathsf{a}_8 \ldots \mathsf{a}_0$ and the rest of the bits hold a $\mathsf{c}$ pattern. The bit pattern propagation of the 4-round distinguisher is shown in Figure 5 in the appendix.

In the first round, the inputs to the ten affected instances of the S-box receive a pair of inputs with a difference of $4_x$. Each input value is repeated $2^9$ times. These S-boxes will, therefore, output a pair of values repeated $2^9$ times. Since

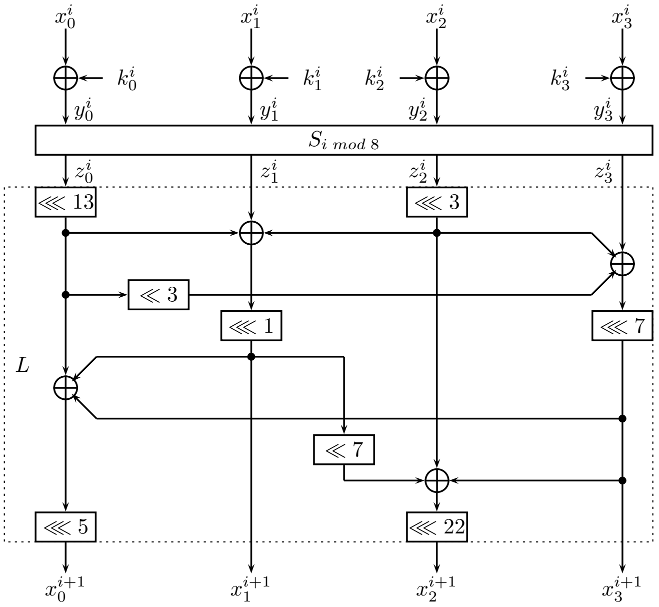

**Fig. 2.** Round function of Serpent in Round $i$

each of the eight S-boxes of Serpent behaves differently to the input difference, the output bits of these S-boxes are denoted as a $\mathsf{d}_i$-pattern. However, at least two bits in each output will hold an $\mathsf{a}_i$-pattern. This is due to one of the design criteria of the S-box, i.e., one-bit input change results in at least two-bit output change. All other bits remain constant. The linear layer in the first round does not affect the balancedness of the structure.

Each instance of the second round S-box may receive a single constant input value, or between two to sixteen different input values. The input values are repeated between $2^6$ and $2^{10}$ times. The number of distinct outputs is therefore even and this ensures that the structure is balanced after the S-box in the second round.

The linear layer in the second round ensures that the number of distinct values in every column occurs an even number times, or that all values occur an odd number of times. All bits will hold a $\mathsf{b}_0^*$-pattern except for a few positions which still retain a $\mathsf{b}_1$-pattern. These bits are then fed to the third round S-box. The number of repetition for each distinct output value matches that of its input and this preserves the balancedness of the structure until after $L$ in the third round.

The application of the fourth round S-boxes is expected to destroy the balancedness of the structure.

**Key Recovery** Note that in the last round $L$ is replaced with key addition, so for four-round Serpent the ciphertexts and the output of the S-boxes in the fourth round is only separated by a simple XOR of the last round key. Hence we only need to guess four bits of the last round key to make use of Equation (1). This attack requires $2 \times 2^{10}$ chosen plaintexts, and it must be repeated 32 times to recover the whole last round key. The time complexity is therefore $2 \times 2^{10} \times 2^4 \times 32 = 2^{20}$ partial decryptions.

In a 5-round attack, we verify Equation (1) three times, for the fourth round S-boxes in bit positions 0,1 and 2. To compute the output of the S-box in position 0, we need to guess 11 bits of $K_4$ and 36 bits of $K_5$. For position 1 we need to guess 32 bits of $K_5$ and 10 bits of $K_4$, and for position 2 we need to guess 20 bits of $K_5$ and 10 more bits of $K_4$. The remaining 40 bits of $K_5$ can then be found by exhaustive search. In order to correctly identify all bits 12 structures are needed, so a total of $12 \times 2^{10} \approx 2^{13.6}$ chosen plaintexts are needed. The time requirement is approximately $(2^{47} + 2^{43} + \ldots + 1) \times 2^{10} \times 3 + 2^{40} \approx 2^{58.7}$ partial decryptions. The memory for the possible key candidates are $2^{44}$ bytes.

The 5-round attack can be extended by adding one round at the beginning. We want the 10 most significant bits of $x_2^1$ to hold the pattern $\mathsf{a}_9\mathsf{a}_8 \ldots \mathsf{a}_0$ and the rest of the bits to hold a $\mathsf{c}$ pattern. If the active bits in $x_2^1$ are traced backwards until the input of the linear transformation of the first round (Round 0), the bits assume the pattern shown in Table 1. Since the number of active S-boxes is 13, we need to guess 52 bits of $K_0$ to find a plaintext structure that will evolve into the desired pattern in $X^1$ when the guess is right. This requires $2^{13.2} \times 2^{52} \approx 2^{65.2}$ chosen plaintexts and approximately $2^{58.7} \times 2^{52} \approx 2^{110.7}$ partial decryptions. The memory requirement is the same as in the 5-round attack.

**Table 1.** Bit patterns of $Z_0$ in the 6-round attack

| Word | Bit Pattern |
|---|---|
| $z_0^0$ | c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c c |
| $z_1^0$ | c c c c c c c c c c c c c c c c c c c c c c $a_9$ $a_8$ $a_7$ $a_6$ $a_5$ $a_4$ $a_3$ $a_2$ $a_1$ $a_0$ |
| $z_2^0$ | $a_2$ $a_1$ $a_0$ c c c c c c c c c c c c c c c c c c c c c c $a_9$ $a_8$ $a_7$ $a_6$ $a_5$ $a_4$ $a_3$ |
| $z_3^0$ | c c c c c c c c c c c c c c c c c c c c c c $a_9$ $a_8$ $a_7$ $a_6$ $a_5$ $a_4$ $a_3$ $a_2$ $a_1$ $a_0$ |

### 3.3 PRESENT

PRESENT [7] is a 64-bit block cipher with key sizes of 80 and 128 bits. It has 31 rounds and is developed exclusively for lightweight applications. Figure 3 illustrates the round function of this cipher. The encryption function is given as:

$$X_{i+1} = L(S(X_i \oplus K_i)), i = 0, 1, \ldots 30$$
$$X_{32} = X_{31} \oplus K_{31}$$

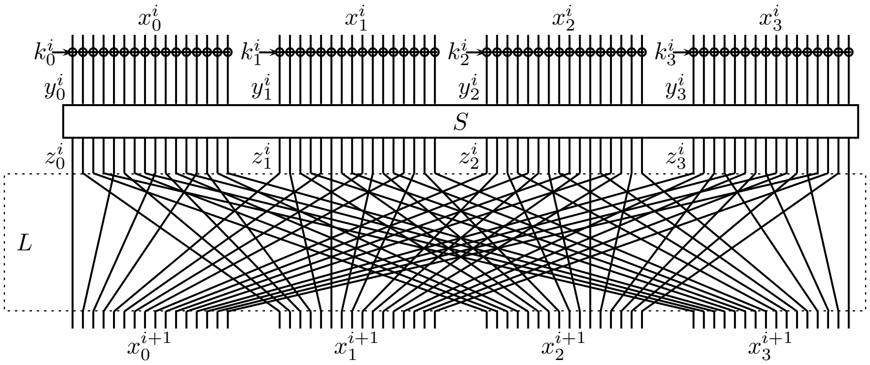

**Fig. 3.** Round function of PRESENT in Round $i$

For consistency with the other analyzed ciphers, the representation of the bits are slightly modified so that the 16 S-boxes in $S$ are implemented in bit-sliced mode like Noekeon and Serpent. The linear layer $L$ is described in Table 2.

**3.5-round Distinguisher** A 3.5-round distinguisher can be built for PRESENT by constructing a structure of $2^4$ chosen plaintexts:

$$(x_0^{0(j)}, x_1^{0(j)}, x_2^{0(j)}, x_3^{0(j)}) = (c_0, c_1, c_2, c_3 \| j)$$

where $c_0, c_1, c_2$ are arbitrary 16-bit constants, $c_3$ are random 12-bit constants and $0 \leq j \leq 15$. The bit propagation of this distinguisher is shown in Figure 6 in the appendix.

**Table 2.** Linear layer $L$ of PRESENT

| Output | Input | | | |
|---|---|---|---|---|
| | $z_0^i$ | $z_1^i$ | $z_2^i$ | $z_3^i$ |
| $x_0^{i+1}$ | 15 11 7 3 | 15 11 7 3 | 15 11 7 3 | 15 11 7 3 |
| $x_1^{i+1}$ | 14 10 6 2 | 14 10 6 2 | 14 10 6 2 | 14 10 6 2 |
| $x_2^{i+1}$ | 13 9 5 1 | 13 9 5 1 | 13 9 5 1 | 13 9 5 1 |
| $x_3^{i+1}$ | 12 8 4 0 | 12 8 4 0 | 12 8 4 0 | 12 8 4 0 |

In the first round, each of the four rightmost S-boxes receives two different input values repeated eight times. The rest of the S-boxes receives only a single constant value repeated sixteen times. The output bits of the four rightmost S-boxes assume the pattern $\mathsf{d}_i\mathsf{d}_i\mathsf{d}_i\mathsf{a}_i$ since there is a differential $1_x \rightarrow w \| 1_x$ occurring with probability 1 where $w \in \{1_x, 3_x, 4_x, 6_x\}$.

In the second round, these sixteen bit patterns are fed to S-boxes 0, 4, 8 and 12. S-box 0 receives the pattern $\mathsf{a}_3\mathsf{a}_2\mathsf{a}_1\mathsf{a}_0$ which represents all possible 4-bit values. S-boxes 4, 8 and 12 receive the pattern $\mathsf{d}_3\mathsf{d}_2\mathsf{d}_1\mathsf{d}_0$. The inputs to the other 12 S-boxes have the pattern $\mathsf{c}$. The input and output values of the active S-boxes are repeated either once or an even number of times, the structure therefore is balanced.

The linear layer in the second round spreads the bits such that each S-box in the third round has the pattern $\mathsf{cccb}_0^*$. Since only one bit position is non-constant, all S-boxes receive at most two different input values. In the preceding round, the output of S-box 0 consists of all possible 4-bit values. Therefore, due to the linear transformation, the number of repetitions for the different input values for S-boxes 0, 4, 8 and 12 in the current round is exactly 8. The output bits of all S-boxes at this point hold the pattern $\mathsf{b}_0^*$ and the structure remains balanced.

In the fourth round, the balancedness of the structure is expected to be destroyed after the application of the S-box.

**Key Recovery** The attack on 4 rounds is exactly the same as for Serpent. The number of chosen plaintexts needed is $2 \times 2^4 = 2^5$ with time complexity of $2 \times 2^4 \times 16 \times 2^4 = 2^{13}$ partial decryptions.

In a 5-round attack, due to the linear layer, the attacker needs to guess an additional $4 \times 4 = 16$ bits of key material from $K_5$, so 5 structures are needed to identify the correct guess. The attack can be repeated 3 times to get 60 bits of $K_5$, the remaining 20 bits of $K_5$ can be found by exhaustive search. The number of chosen plaintexts needed is $5 \times 2^4 \approx 2^{6.4}$, and the time complexity is $(2^{20} + 2^{16} + \ldots + 1) \times 2^4 \times 3 + 2^{20} \approx 2^{25.7}$ partial decryptions. The memory requirement is small.

A 6-round attack can be made by adding one round at the beginning to construct a 4.5-round distinguisher. The plaintexts are chosen such that the inputs into the second round assume the pattern of the inputs of the 3.5-round distinguisher described above. There are four active S-boxes in the first round, and hence 16 bits of $K_0$ needs to be guessed. This 6-round attack would require $2^{16} \times 2^{6.4} \approx 2^{22.4}$ chosen plaintexts and $2^{16} \times 2^{25.7} \approx 2^{41.7}$ partial decryptions. The memory complexity is still small.

We can extend the attack to seven rounds by adding even another round in the end, but this attack is only better than exhaustive search for 128-bit keys. In a 7-round attack, the whole 64 bits of $K_7$ is needed to be guessed. After examining the key schedule for 128-bit keys, we find that 3 bits of $K_6$ and 58 bits of $K_5$ are given from guessing all of $K_7$. These known bits overlap in one of the bits needed from $K_6$ and three of the bits needed from $K_5$, so in total we need to guess $1 + 15 + 64 = 80$ bits of key material. The attack requires $20 \times 2^{16} \times 2^4 \approx 2^{24.3}$ chosen plaintexts and $(2^{80} + 2^{76} + \ldots + 1) \times 2^4 \times 2^{16} \approx 2^{100.1}$ partial decryptions. A total of $2^{80}$ bits are required to keep track of possible values for the 80 key bits, so the memory complexity is $2^{77}$ bytes.

### 3.4 Summary

The complexities of key recovery attacks on Noekeon, Serpent and PRESENT depend largely on the linear component of the round function. All 4-round attacks have been implemented on a single desktop PC. The attacks took only a few seconds to recover the last round subkey. A summary of attacks presented in this paper is shown in Table 3.

**Table 3.** Summary of attacks

| Cipher | Rounds | Complexity | | |
|---|---|---|---|---|
| | | Data | Time | Memory |
| Noekeon | 4 | $2^{17}$ CP | $2^{26}$ | small |
| | 5 | $2^{20.6}$ CP | $2^{108.1}$ | $2^{89}$ bytes |
| Serpent | 4 | $2^{11}$ CP | $2^{20}$ | small |
| | 5 | $2^{13.6}$ CP | $2^{58.7}$ | $2^{44}$ bytes |
| | 6 | $2^{65.2}$ CP | $2^{110.7}$ | $2^{44}$ bytes |
| PRESENT | 4 | $2^{5}$ CP | $2^{13}$ | small |
| | 5 | $2^{6.4}$ CP | $2^{25.7}$ | small |
| | 6 | $2^{22.4}$ CP | $2^{41.7}$ | small |
| | 7 | $2^{24.3}$ CP | $2^{100.1}$ | $2^{77}$ bytes |

## 4 Related Work

The applicability of the integral attack on bit-oriented ciphers was mentioned in Knudsen and Wagner's work [11]. The attack is demonstrated on the Data Encryption Standard (DES). The attack, however, works only for a very few rounds of the DES. Lucks [13] also attacked Twofish, which is not a purely byte-based cipher, with integral cryptanalysis. In Piret's thesis [14, pg 79-82], the construction of an integral distinguisher for Serpent was discussed. The distinguisher, however, does not occur with certainty and the number of rounds of the distinguisher was not explicitly mentioned.

In another work, Biryukov and Shamir [6] show how to attack a generic cipher structure which consists of non-linear and linear layers which are unknown. The technique, called the multiset attack, makes use of several multiset properties. These properties take into account whether the multiset: (1) contains arbitrary repetitions of a single value; (2) takes on all possible values; (3) contains values which occur an even number of times; (4) XOR sum equals 0; (5) has either property (2) or (3). Therefore, there is some similarities to the notations described in our work.

## 5 Discussion and Conclusion

In this paper, we examined the integral attack using a bit-pattern based approach. It differs from classical integral cryptanalysis in that the order of the texts in a structure becomes important, and gives the cryptanalyst a more refined notation for the texts in the structure. This information allows an attacker to gain a detailed analysis of the individual bit that propagates through the rounds. This is especially useful in analyzing the attack on ciphers that have bit-oriented round functions.

In the Noekeon document [9], it is stated that *there are no 4-round differential trails with a predicted prop ratio above* $2^{-48}$. A prop ratio is the fraction of input pairs with a fixed difference that propagates into a fixed output difference. The Appendix of the Noekeon document describes that the differential trail propagates until before the non-linear component in the fourth round. In this paper, a 3.5-round distinguisher with probability 1 is discovered in which the balancedness of the structure can be retained until just before the S-box layer in the fourth round. This distinguisher is therefore comparable to the differential trail described in the document. Since our distinguisher can be constructed using just $2^{16}$ chosen plaintexts (as opposed to the differential attack which requires on the order of $2^{48}$ plaintexts), attacks using this distinguisher are much more efficient than using the best differential trail described by the authors.

The designers of Serpent have shown in their submission to the AES competition [1] that the probability of the best 5-round differential characteristic is less than $2^{-42}$, so a differential attack on 5-round Serpent would require on the order of $2^{42}$ chosen plaintexts. The 5-round bit-pattern based integral attack described in this paper requires only $2^{13.6}$ chosen plaintexts. It should be noted that there exists better attacks on Serpent than what is reported here, the best we are aware of is a linear attack that breaks 10 rounds of Serpent with 128-bit keys [8]. The complexities of the six-round differential attack reported in [12] are comparable to the bit-pattern based integral attack described here.

In the PRESENT document [7] resistance to integral cryptanalysis is explained by noting that integral attacks are suited for ciphers with a word-wise structure, and that the design of PRESENT is bitwise. We have shown here that bit-pattern based integral cryptanalysis is indeed suited for PRESENT. Moreover, the authors show that a differential characteristic over five rounds has probability at most $2^{-20}$, so a differential attack on five rounds would need on the order of $2^{20}$ plaintexts to succeed. In contrast, bit-pattern based integral cryptanalysis breaks five-round PRESENT with 80 chosen plaintexts.

Over a few rounds, bit-pattern based integral cryptanalysis of the three ciphers studied here is comparable to differential cryptanalysis in time complexity, but require in general much less chosen plaintext. However, differential cryptanalysis is easy to extend to more rounds whereas integral cryptanalysis can not be extended beyond a certain point. Even though the attacks do not pose a serious threat to the ciphers presented in this paper, it shows that the integral attack can still be applied to bit-oriented ciphers.

## Acknowledgements

We would like to thank Adi Shamir for pointing out a related work in [6]. We also thank Leonie Simpson and anonymous referees that helped to improve this paper.

## References

1. Anderson, R., Biham, E., Knudsen, L.: Serpent: A Proposal for the Advanced Encryption Standard. In: NIST AES Proposal (1998), available at `http://www.cl.cam.ac.uk/~rja14/serpent.html`
2. Biham, E., Dunkelman, O., Keller, N.: The Rectangle Attack – Rectangling the Serpent. In: Pfitzmann, B. (ed.) Eurocrypt 2001. LNCS, vol. 2045, pp. 340–357. Springer, Heidelberg (2001)
3. Biham, E., Dunkelman, O., Keller, N.: Linear Cryptanalysis of Reduced Round Serpent. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 16–27. Springer, Heidelberg (2002)
4. Biham, E., Dunkelman, O., Keller, N.: New Results on Boomerang and Rectangle Attacks. In: Daemen, J., Rijmen, V. (eds.) FSE 2002. LNCS, vol. 2365, pp. 1–16. Springer, Heidelberg (2002)
5. Biham, E., Dunkelman, O., Keller, N.: Differential-Linear Cryptanalysis of Serpent. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 9–21. Springer, Heidelberg (2003)
6. Biryukov, A., Shamir, A.: Structural Cryptanalysis of SASAS. In: Pfitzmann, B. (ed.) Eurocrypt 2001. LNCS, vol. 2045, pp. 394–405. Springer, Heidelberg (2001)
7. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I., (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
8. Collard, B., Standaert, F.-X., Quisquater, J.-J.: Improved and Multiple Linear Cryptanalysis of Reduced Round Serpent. In: Inscrypt 2007. LNCS, to appear (2007)
9. Daemen, J., Peeters, M., Van Assche, G., Rijmen, V.: Nessie Proposal: NOEKEON. In: First Open NESSIE Workshop (2000), available at `http://gro.noekeon.org/`
10. Kelsey, J., Kohno, T., Schneier, B.: Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 75–93. Springer, Heidelberg (2001)
11. Knudsen, L., Wagner, D.: Integral Cryptanalysis. In: Daeman, J., Rijmen, V., (eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002)
12. Kohno, T., Kelsey, J., Schneier, B.: Preliminary Cryptanalysis of Reduced-Round Serpent. In: The Third Advanced Encryption Standard Candidate Conference, pp. 195–211. NIST (2000), available at `http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/aes3conf.htm`
13. Lucks, S.: The Saturation Attack – A Bait for Twofish. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 1–15. Springer, Heidelberg (2002)
14. Piret, G.: Block Ciphers: Security Proofs, Cryptanalysis, Design, and Fault Attacks. PhD Thesis, Université Catholique de Louvain (2005), available at `http://www.di.ens.fr/~piret/`

# Appendix

**Table 4.** Example of bit pattern values in a $2^4$ structure

| $\mathsf{c}$ | | $\mathsf{a}_3$ | | $\mathsf{a}_2$ | | $\mathsf{a}_1$ | | $\mathsf{a}_0$ | | $\mathsf{b}_2$ | | $\mathsf{b}_1$ | | | | $\mathsf{b}_0^*$ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | $\mathsf{a}_3 \oplus \mathsf{a}_2$ | | $\mathsf{a}_3 \oplus \mathsf{a}_2 \oplus \mathsf{a}_1$ | | | | $\mathsf{a}_2 \oplus \mathsf{a}_0$ | | $\mathsf{a}_3 \oplus \mathsf{a}_1 \oplus \mathsf{a}_0$ | | | |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Table 4 depicts the possible values of bit patterns in a $2^4$ structure. Each individual pattern has two columns to indicate that a pattern may start with bit value '0' or bit value '1'. Recall that the pattern $\mathsf{b}_0^*$ and $\mathsf{b}_i$ where $i > 0$ are not restricted to hold the values of only the XOR combinations of $\mathsf{a}_i$ patterns. If the patterns are composed of XOR combinations of $\mathsf{a}_i$ patterns, the number of occurrences of bit '0' is the same as bit '1'.

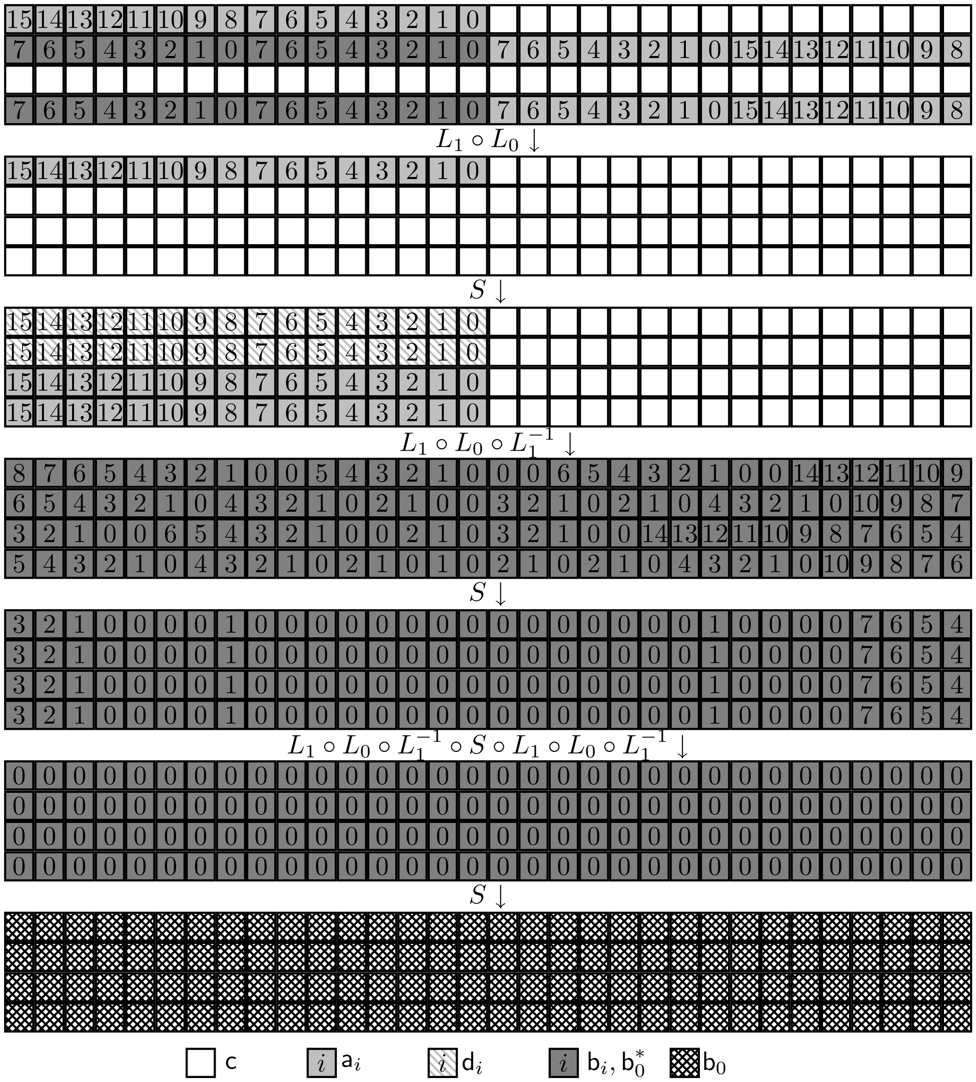
$L_1 \circ L_0 \downarrow$
$S \downarrow$
$L_1 \circ L_0 \circ L_1^{-1} \downarrow$
$S \downarrow$
$L_1 \circ L_0 \circ L_1^{-1} \circ S \circ L_1 \circ L_0 \circ L_1^{-1} \downarrow$
$S \downarrow$
c
$\mathsf{a}_i$
$\mathsf{d}_i$
$\mathsf{b}_i, \mathsf{b}_0^*$
$\mathsf{b}_0$

**Fig. 4.** The 3.5-round integral distinguisher for Noekeon. Top row of cipher block is word 0, bottom row is word 3. The rightmost column corresponds to the least significant bit (bit 0) in each word.

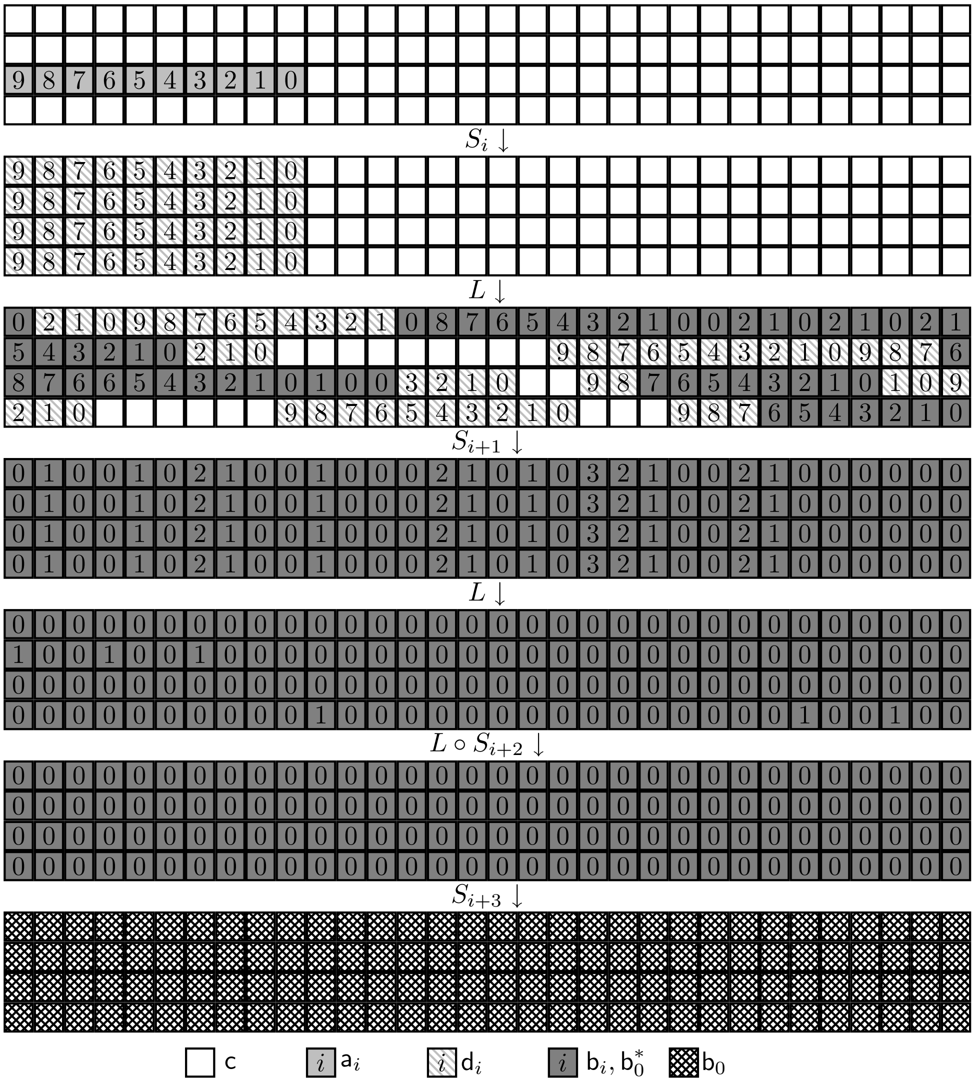

**Fig. 5.** The 3.5-round integral distinguisher for Serpent. Top row of cipher block is word 0, bottom row is word 3. The rightmost column corresponds to the least significant bit (bit 0) in each word.

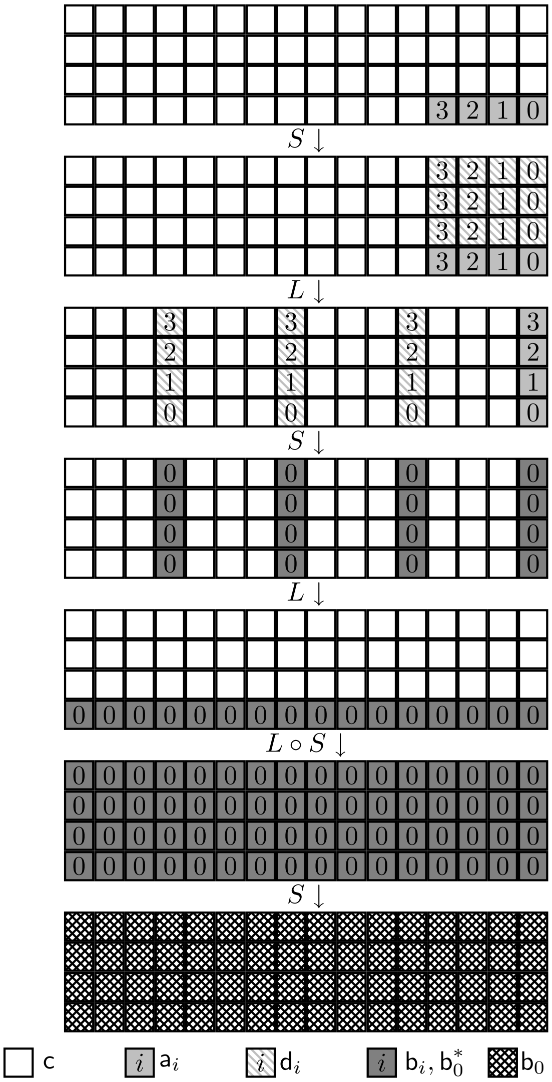

**Fig. 6.** The 3.5-round integral distinguisher for PRESENT. Top row of cipher block is word 0, bottom row is word 3. The rightmost column corresponds to the least significant bit (bit 0) in each word.