Bit-Pattern Based Integral Attack

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

[1] Information Security Institute, Queensland University of Technology, Australia
[2] Selmersenteret , University of Bergen, Norway
[3] Institute for Infocomm Research, A*STAR, Singapore

Outline

1. Introduction
2. Bit-Pattern Based Integral Attack
   - Background
   - Type of Block Ciphers
   - Attack Algorithm
3. Application to Block Ciphers
   - PRESENT
   - Noekeon
   - Serpent
4. Discussion and Conclusion

Integral attack is very suitable for word-oriented ciphers

Problem for bit-oriented ciphers

Any all-values property (a set of all possible values) of bijective S-box output will be destroyed by bit-wise linear component

Solution

View each input bit position within structure as a sequence of bit patterns – bit-pattern based integral attack

Application of bit-pattern based integral attack

7-round PRESENT, 5-round Noekeon and 6-round Serpent

Integral attack is very suitable for word-oriented ciphers

Problem for bit-oriented ciphers

Any all-values property (a set of all possible values) of bijective S-box output will be destroyed by bit-wise linear component

Solution

View each input bit position within structure as a sequence of bit patterns – bit-pattern based integral attack

Application of bit-pattern based integral attack

7-round PRESENT, 5-round Noekeon and 6-round Serpent

Introduction

Integral attack is very suitable for word-oriented ciphers

Problem for bit-oriented ciphers

Any all-values property (a set of all possible values) of bijective S-box output will be destroyed by bit-wise linear component

Solution

View each input bit position within structure as a sequence of bit patterns – bit-pattern based integral attack

Application of bit-pattern based integral attack

7-round PRESENT, 5-round Noekeon and 6-round Serpent

Outline

1 Introduction

2 Bit-Pattern Based Integral Attack
- Background
- Type of Block Ciphers
- Attack Algorithm

3 Application to Block Ciphers
- PRESENT
- Noekeon
- Serpent

4 Discussion and Conclusion

Bit-Pattern Based Integral Attack

Comparison between conventional byte-based and bit-based integral attacks

Conventional

- A set of m-bit active words into a single S-box in Round 1
- The active S-box receives all possible 2^m values in Round 1
- Inputs form an unordered set

Bit-Pattern Based

- A set of m-bit active words into m S-boxes in Round 1
- Each active s-box receives a pair of values in Round 1
- Inputs form an ordered set

Bit-Pattern Based Integral Attack

Comparison between conventional byte-based and bit-based integral attacks

Conventional

- A set of m-bit active words into a single S-box in Round 1
- The active S-box receives all possible 2^m values in Round 1
- Inputs form an unordered set

Bit-Pattern Based

- A set of m-bit active words into m S-boxes in Round 1
- Each active s-box receives a pair of values in Round 1
- Inputs form an ordered set

Bit-Pattern Based Integral Attack

Comparison between conventional byte-based and bit-based integral attacks

Conventional

- A set of m-bit active words into a single S-box in Round 1
- The active S-box receives all possible 2^m values in Round 1
- Inputs form an unordered set

Bit-Pattern Based

- A set of m-bit active words into m S-boxes in Round 1
- Each active s-box receives a pair of values in Round 1
- Inputs form an ordered set

Notations

Each bit position within structure is treated independently. The possible patterns:

- Constant **c**: the bits consist of only bit ‘0’ or ‘1’.
  E.g. 00000000 or 11111111
- Active $\mathbf{a}_i$: alternating blocks of 2^i consecutive bits (‘0’ and ‘1’).
  E.g. a_1: 00110011
- Balance $\mathbf{b}_i$: repetition of blocks of 2^i consecutive bits (‘0’ and ‘1’) – not alternating.
  E.g. b_1: 00111100
  E.g. b_0^*: 10110001
  E.g. b_0: 10000000
- Dual $\mathbf{d}_i$: c or a_i

Balancedness

Balanced pattern: XOR sum = 0
For b_0, b_0^* is balanced, b_0 is not necessarily balanced

Notations

Each bit position within structure is treated independently. The possible patterns:

- Constant **c**: the bits consist of only bit '0' or '1'.
  E.g. 00000000 or 11111111
- Active $\mathbf{a}_i$: alternating blocks of 2^i consecutive bits ('0' and '1').
  E.g. a_1: 00110011
- Balance $\mathbf{b}_i$: repetition of blocks of 2^i consecutive bits ('0' and '1') – not alternating.
  E.g. b_1: 00111100
  E.g. b_0^*: 10110001
  E.g. b_0: 10000000
- Dual $\mathbf{d}_i$: c or a_i

Balancedness

Balanced pattern: XOR sum = 0
For b_0, b_0^* is balanced, b_0 is not necessarily balanced

Notations

Each bit position within structure is treated independently. The possible patterns:

- Constant **c**: the bits consist of only bit '0' or '1'.
  E.g. 00000000 or 11111111
- Active **a_i**: alternating blocks of 2^i consecutive bits ('0' and '1').
  E.g. a_1: 00110011
- Balance **b_i**: repetition of blocks of 2^i consecutive bits ('0' and '1') – not alternating.
  E.g. b_1: 00111100
  E.g. b_0^*: 10110001
  E.g. b_0: 10000000
- Dual **d_i**: c or a_i

Balancedness

Balanced pattern: XOR sum = 0
For b_0, b_0^* is balanced, b_0 is not necessarily balanced

Notations

Each bit position within structure is treated independently. The possible patterns:

- Constant **c**: the bits consist of only bit '0' or '1'.
  E.g. 00000000 or 11111111
- Active **a_i**: alternating blocks of 2^i consecutive bits ('0' and '1').
  E.g. a_1: 00110011
- Balance **b_i**: repetition of blocks of 2^i consecutive bits ('0' and '1') – not alternating.
  E.g. b_1: 00111100
  E.g. b_0^*: 10110001
  E.g. b_0: 10000000
- Dual **d_i**: c or a_i

Balancedness

Balanced pattern: XOR sum = 0
For b_0, b_0^* is balanced, b_0 is not necessarily balanced

Notations

Each bit position within structure is treated independently. The possible patterns:

- Constant **c**: the bits consist of only bit '0' or '1'.
  E.g. 00000000 or 11111111
- Active **a_i**: alternating blocks of 2^i consecutive bits ('0' and '1').
  E.g. a_1: 00110011
- Balance **b_i**: repetition of blocks of 2^i consecutive bits ('0' and '1') – not alternating.
  E.g. b_1: 00111100
  E.g. b_0^*: 10110001
  E.g. b_0: 10000000
- Dual **d_i**: c or a_i

Balancedness

Balanced pattern: XOR sum = 0

For b_0, b_0^* is balanced, b_0 is not necessarily balanced

Notations

Each bit position within structure is treated independently. The possible patterns:

- Constant **c**: the bits consist of only bit '0' or '1'.
  E.g. 00000000 or 11111111
- Active $\mathbf{a}_i$: alternating blocks of 2^i consecutive bits ('0' and '1').
  E.g. a_1: 00110011
- Balance $\mathbf{b}_i$: repetition of blocks of 2^i consecutive bits ('0' and '1') – not alternating.
  E.g. b_1: 00111100
  E.g. b_0^*: 10110001
  E.g. b_0: 10000000
- Dual $\mathbf{d}_i$: c or a_i

Balancedness

Balanced pattern: XOR sum = 0
For b_0, b_0^* is balanced, b_0 is not necessarily balanced

Example

Example of bit patterns in a 4-bit structure

c	a_2	b_0^*	b_1	Hex
0	1	0	1	5
0	1	1	1	7
0	1	0	0	4
0	1	1	0	6
0	0	1	0	2
0	0	0	0	0
0	0	1	1	3
0	0	0	1	1
0	1	0	0	4
0	1	1	0	6
0	1	0	1	5
0	1	1	1	7
0	0	1	1	3
0	0	0	1	1
0	0	1	0	2
0	0	0	0	0

Note: $b_0^* = a_2 \oplus a_0$, $b_1 = a_3 \oplus a_2 \oplus a_1$

Tracing Bit Patterns

Operations on bit patterns

- $c \oplus c = c$
- $a_i \oplus a_i = c$
- $a_i \oplus c = a_i$
- $a_i \oplus a_j = b_j$ for $j < i$
- $b_i \oplus b_j = b_j$ for $j < i$
- $p \oplus b_0^* = b_0^*$ for $p \in \{a, b\}$ and $p \neq b_0$

S-Box output patterns

Every output bit position will have a b_i pattern where i is the smallest index found in the input patterns

Example

c	$\oplus$	c	c
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1
0		1	1

Tracing Bit Patterns

Operations on bit patterns

- $c \oplus c = c$
- $a_i \oplus a_i = c$
- $a_i \oplus c = a_i$
- $a_i \oplus a_j = b_j$ for $j < i$
- $b_i \oplus b_j = b_j$ for $j < i$
- $p \oplus b_0^* = b_0^*$ for $p \in \{a, b\}$ and $p \neq b_0$

S-Box output patterns

Every output bit position will have a b_i pattern where i is the smallest index found in the input patterns

Example

a_2	$\oplus$	a_2	c
0		1	**1**
0		1	**1**
0		1	**1**
0		1	**1**
1		0	**1**
1		0	**1**
1		0	**1**
1		0	**1**
0		1	**1**
0		1	**1**
0		1	**1**
0		1	**1**
1		0	**1**
1		0	**1**
1		0	**1**
1		0	**1**

Tracing Bit Patterns

Operations on bit patterns

- $c \oplus c = c$
- $a_i \oplus a_i = c$
- $a_i \oplus c = a_i$
- $a_i \oplus a_j = b_j$ for $j < i$
- $b_i \oplus b_j = b_j$ for $j < i$
- $p \oplus b_0^* = b_0^*$ for $p \in \{a, b\}$ and $p \neq b_0$

S-Box output patterns

Every output bit position will have a b_i pattern where i is the smallest index found in the input patterns

Example

a_2	$\oplus$	c	a_2
0		0	0
0		0	0
0		0	0
0		0	0
1		0	1
1		0	1
1		0	1
1		0	1
0		0	0
0		0	0
0		0	0
0		0	0
1		0	1
1		0	1
1		0	1
1		0	1

Tracing Bit Patterns

Operations on bit patterns

- $c \oplus c = c$
- $a_i \oplus a_i = c$
- $a_i \oplus c = a_i$
- $a_i \oplus a_j = b_j$ for $j < i$
- $b_i \oplus b_j = b_j$ for $j < i$
- $p \oplus b_0^* = b_0^*$ for $p \in \{a, b\}$ and $p \neq b_0$

S-Box output patterns

Every output bit position will have a b_i pattern where i is the smallest index found in the input patterns

Example

a_3	$\oplus$	a_2	b_2
0		1	1
0		1	1
0		1	1
0		1	1
0		0	0
0		0	0
0		0	0
0		0	0
1		1	0
1		1	0
1		1	0
1		1	0
1		0	1
1		0	1
1		0	1
1		0	1

Tracing Bit Patterns

Operations on bit patterns

- $c \oplus c = c$
- $a_i \oplus a_i = c$
- $a_i \oplus c = a_i$
- $a_i \oplus a_j = b_j$ for $j < i$
- $b_i \oplus b_j = b_j$ for $j < i$
- $p \oplus b_0^* = b_0^*$ for $p \in \{a, b\}$ and $p \neq b_0$

Example

S-Box output patterns

Every output bit position will have a b_i pattern where i is the smallest index found in the input patterns

Tracing Bit Patterns

Operations on bit patterns

- $c \oplus c = c$
- $a_i \oplus a_i = c$
- $a_i \oplus c = a_i$
- $a_i \oplus a_j = b_j$ for $j < i$
- $b_i \oplus b_j = b_j$ for $j < i$
- $p \oplus b_0^* = b_0^*$ for $p \in \{a, b\}$ and $p \neq b_0$

S-Box output patterns

Every output bit position will have a b_i pattern where i is the smallest index found in the input patterns

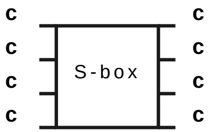
Example

Tracing Bit Patterns

Example of s-box input/output bit patterns

- a set of constant inputs
- a set of all possible inputs
- a set of distinct inputs which occur an even (odd) number of times

1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)

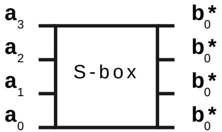


Tracing Bit Patterns

Example of s-box
input/output bit patterns

- a set of constant inputs
- **a set of all possible inputs**
- a set of distinct inputs which occur an even (odd) number of times

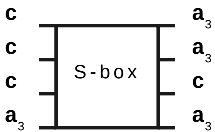
0000 (0)	1100 (C)
0001 (1)	0101 (5)
0010 (2)	0110 (6)
0011 (3)	1011 (B)
0100 (4)	1001 (9)
0101 (5)	0000 (0)
0110 (6)	1010 (A)
0111 (7)	1101 (D)
1000 (8)	0011 (3)
1001 (9)	1110 (E)
1010 (A)	1111 (F)
1011 (B)	1000 (8)
1100 (C)	0100 (4)
1101 (D)	0111 (7)
1110 (E)	0001 (1)
1111 (F)	0010 (2)



Example of s-box input/output bit patterns

- a set of constant inputs
- a set of all possible inputs
- a set of distinct inputs which occur an even (odd) number of times

Input	Output
1000 (8)	0011 (3)
1000 (8)	0011 (3)
1000 (8)	0011 (3)
1000 (8)	0011 (3)
1000 (8)	0011 (3)
1000 (8)	0011 (3)
1000 (8)	0011 (3)
1000 (8)	0011 (3)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)
1001 (9)	1110 (E)



How many distinct inputs into an S-box?

Lemma 1

- Given a set of input patterns $I_0, \ldots, I_{m-1}$ to an $m \times m$ bijective S-box, expressed as linear combinations of a_i-patterns where $a = (a_0, \ldots, a_{m-1})^T$
  - Let $\mathbf{I} = (I_0, \ldots, I_{m-1})^T$
  - Represent as product of matrix $Ma = \mathbf{I}$
  - The number of distinct values to S-box = $2^{rank(M)}$

Use to determine whether balancedness of structure is retained or not after S-box

Example

Example

- Input: $(l_0, l_1, l_2, l_3) = (a_0, a_2, b_0^*, b_1)$
- Matrix:

$$Ma = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix}$$

- Rank(M): 3
- Number of distinct input values: 2^3

Sample values

a_0	a_2	b_0^*	b_1	Hex
0	1	0	1	5
1	1	1	1	F
0	1	0	0	4
1	1	1	0	E
0	0	1	0	2
1	0	0	0	8
0	0	1	1	3
1	0	0	1	9
0	1	0	0	4
1	1	1	0	E
0	1	0	1	5
1	1	1	1	F
0	0	1	1	3
1	0	0	1	9
0	0	1	0	2
1	0	0	0	8

Note: $b_0^* = a_2 \oplus a_0$, $b_1 = a_3 \oplus a_2 \oplus a_1$

Outline

1 Introduction

2 Bit-Pattern Based Integral Attack
- Background
- Type of Block Ciphers
- Attack Algorithm

3 Application to Block Ciphers
- PRESENT
- Noekeon
- Serpent

4 Discussion and Conclusion

Generic Structure of Block Cipher Round Function

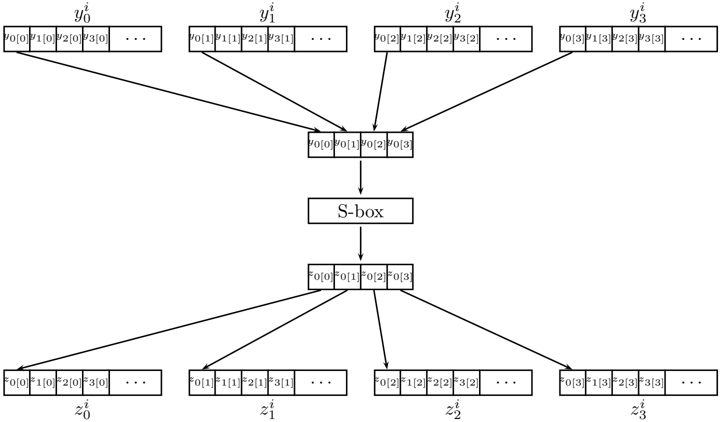
Attack applies to block cipher which can be represented by:

$$
\begin{tikzcd}
X_i = (x_0^i, x_1^i, x_2^i, x_3^i) \arrow[d] \\
\boxed{\quad L_0 \quad} \arrow[d, "Y_i = (y_0^i, y_1^i, y_2^i, y_3^i)"] \\
\boxed{\quad S \quad} \arrow[d, "Z_i = (z_0^i, z_1^i, z_2^i, z_3^i)"] \\
\boxed{\quad L_1 \quad} \arrow[d] \\
X_{i+1} = (x_0^i, x_1^i, x_2^i, x_3^i)
\end{tikzcd}
$$

The analyzed ciphers (PRESENT, Noekeon and Serpent) fit this general structure and the non-linear layer is composed of bijective S-boxes

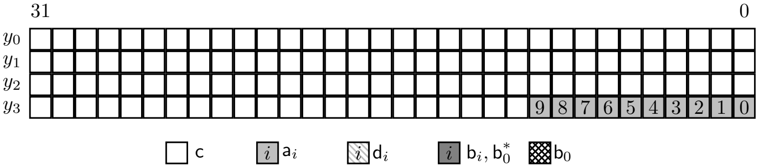
Generic Structure of Block Cipher Round Function

Example of 4 $\times$ 4 bijective S-box



Generic Structure of Block Cipher Round Function

Representation of input bits into round function



Each row represents a sub-block
Each column represents input into a single S-box

Outline

1. Introduction
2. Bit-Pattern Based Integral Attack
   - Background
   - Type of Block Ciphers
   - Attack Algorithm
3. Application to Block Ciphers
   - PRESENT
   - Noekeon
   - Serpent
4. Discussion and Conclusion

Bit-Pattern Based Integral Attack Algorithm

1. Precomputation
   1. Analyze round function to identify distinguisher
2. Construct a structure of plaintexts that matches distinguisher
3. Encrypt all plaintexts in structure and get corresponding ciphertexts
4. Initialize an array $A[]$ of size 2^k bits with all '1's
5. Set $v = 0$
6. While # of entries such that $A[v] = 1$ is greater than one
   1. Partially decrypt all ciphertexts using the value v as partial subkey bits to find the output bits of one S-box in round r
   2. If Equation (1): $\bigoplus_{j=0}^{m-1} Y_r^{(j)} \neq 0$, set $A[v] = 0$
   3. $v = v + 1$
7. Output value v for which $A[v] = 1$ as correct subkey bits

Bit-Pattern Based Integral Attack Algorithm

1. Precomputation
   1. Analyze round function to identify distinguisher
2. Construct a structure of plaintexts that matches distinguisher
3. Encrypt all plaintexts in structure and get corresponding ciphertexts
4. Initialize an array $A[]$ of size 2^k bits with all '1's
5. Set $v = 0$
6. While # of entries such that $A[v] = 1$ is greater than one
   1. Partially decrypt all ciphertexts using the value v as partial subkey bits to find the output bits of one S-box in round r
   2. If Equation (1): $\bigoplus_{j=0}^{m-1} Y_r^{(j)} \neq 0$, set $A[v] = 0$
   3. $v = v + 1$
7. Output value v for which $A[v] = 1$ as correct subkey bits

Bit-Pattern Based Integral Attack Algorithm

1. Precomputation
   1. Analyze round function to identify distinguisher
2. Construct a structure of plaintexts that matches distinguisher
3. Encrypt all plaintexts in structure and get corresponding ciphertexts
4. Initialize an array $A[]$ of size 2^k bits with all '1's
5. Set $v = 0$
6. While # of entries such that $A[v] = 1$ is greater than one
   1. Partially decrypt all ciphertexts using the value v as partial subkey bits to find the output bits of one S-box in round r
   2. If Equation (1): $\bigoplus_{j=0}^{m-1} Y_r^{(j)} \neq 0$, set $A[v] = 0$
   3. $v = v + 1$
7. Output value v for which $A[v] = 1$ as correct subkey bits

Bit-Pattern Based Integral Attack Algorithm

1. Precomputation
    1. Analyze round function to identify distinguisher
2. Construct a structure of plaintexts that matches distinguisher
3. Encrypt all plaintexts in structure and get corresponding ciphertexts
4. Initialize an array $A[]$ of size 2^k bits with all '1's
5. Set $v = 0$
6. While # of entries such that $A[v] = 1$ is greater than one
    1. Partially decrypt all ciphertexts using the value v as partial subkey bits to find the output bits of one S-box in round r
    2. If Equation (1): $\bigoplus_{j=0}^{m-1} Y_r^{(j)} \neq 0$, set $A[v] = 0$
    3. $v = v + 1$
7. Output value v for which $A[v] = 1$ as correct subkey bits

Bit-Pattern Based Integral Attack Algorithm

1. Precomputation
   1. Analyze round function to identify distinguisher
2. Construct a structure of plaintexts that matches distinguisher
3. Encrypt all plaintexts in structure and get corresponding ciphertexts
4. Initialize an array $A[]$ of size 2^k bits with all '1's
5. Set $v = 0$
6. While # of entries such that $A[v] = 1$ is greater than one
   1. Partially decrypt all ciphertexts using the value v as partial subkey bits to find the output bits of one S-box in round r
   2. If Equation (1): $\bigoplus_{j=0}^{m-1} Y_r^{(j)} \neq 0$, set $A[v] = 0$
   3. $v = v + 1$
7. Output value v for which $A[v] = 1$ as correct subkey bits

Outline

1. Introduction
2. Bit-Pattern Based Integral Attack
   - Background
   - Type of Block Ciphers
   - Attack Algorithm
3. Application to Block Ciphers
   - PRESENT
   - Noekeon
   - Serpent
4. Discussion and Conclusion

PRESENT Block Cipher

Specification [Bogdanov et al., 2007]

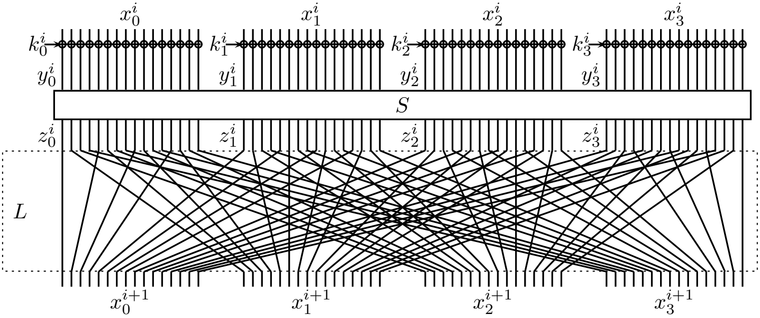
- Block size: 64 bits
- Key size: 80, 128 bits
- Number of Rounds: 31
- Encryption

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

Attacks

- Outlined by designers (differential, linear, integral attack – not suitable, algebraic, key schedule)
- Bit-pattern based integral attack

Round Function

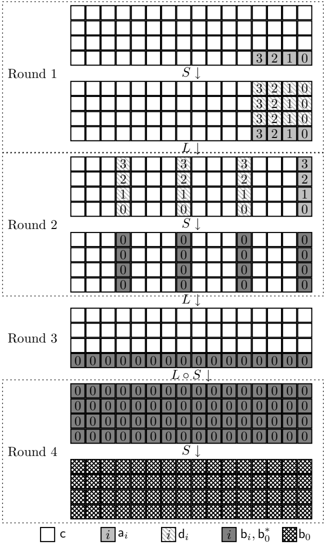


Precomputation

- S-box is 4×4 bijective
- Differential $1_x \to w \| 1_x$ with probability 1, $w \in \{1_x, 3_x, 4_x, 6_x\}$
- Linear layer L is a simple permutation

Bit-Pattern Based Integral Attack on PRESENT

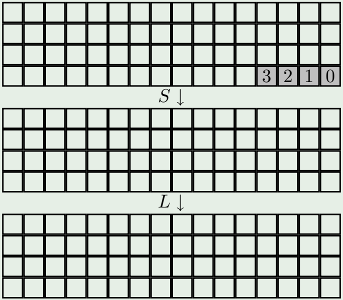
3.5-round distinguisher



Round 1

- Prepare a structure of 2^4 plaintexts
- Active S-boxes
  - Input: 2 distinct values repeated 8 times
  - Output: 2 distinct values repeated 8 times
- Linear

Distinguisher

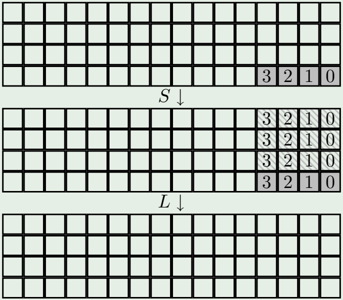


Bit-Pattern Based Integral Attack on PRESENT

Round 1

- Prepare a structure of 2^4 plaintexts
- Active S-boxes
  - Input: 2 distinct values repeated 8 times
  - Output: 2 distinct values repeated 8 times
- Linear

Distinguisher

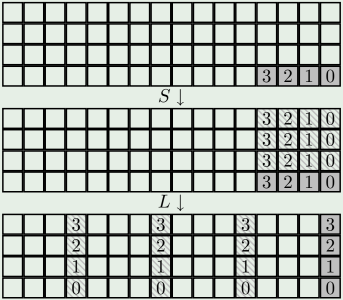


Bit-Pattern Based Integral Attack on PRESENT

Round 1

- Prepare a structure of 2^4 plaintexts
- Active S-boxes
  - Input: 2 distinct values repeated 8 times
  - Output: 2 distinct values repeated 8 times
- Linear

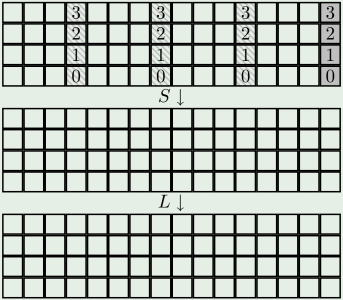
Distinguisher



Round 2

- Active S-boxes
  - S-box 0 input: 16 distinct values each repeated once
  - S-box 4,8,12 input: 1/2/4/8/16 distinct values
  - S-box 0 output: 16 distinct values each repeated once
  - S-box 4,8,12 output: 1/2/4/8/16 distinct values
- Linear

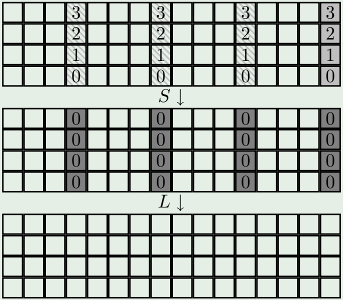
Distinguisher



Round 2

- Active S-boxes
  - S-box 0 input: 16 distinct values each repeated once
  - S-box 4,8,12 input: 1/2/4/8/16 distinct values
  - S-box 0 output: 16 distinct values each repeated once
  - S-box 4,8,12 output: 1/2/4/8/16 distinct values
- Linear

Distinguisher

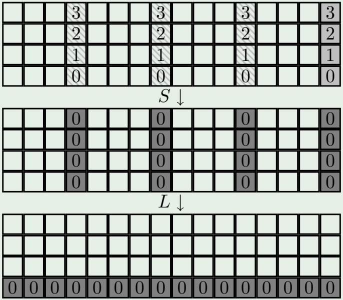


Bit-Pattern Based Integral Attack on PRESENT

Round 2

- Active S-boxes
  - S-box 0 input: 16 distinct values each repeated once
  - S-box 4,8,12 input: 1/2/4/8/16 distinct values
  - S-box 0 output: 16 distinct values each repeated once
  - S-box 4,8,12 output: 1/2/4/8/16 distinct values
- Linear

Distinguisher

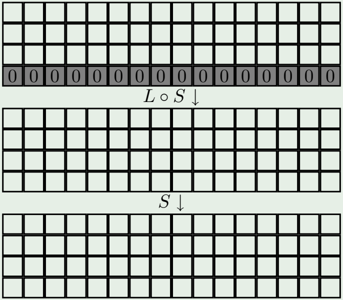


Bit-Pattern Based Integral Attack on PRESENT

Rounds 3 and 4

- Active S-boxes
  - Input: 1/2 distinct values repeated even number of times
  - Output: 1/2 distinct values repeated even number of times
- Linear
- S-box destroys balancedness

Distinguisher

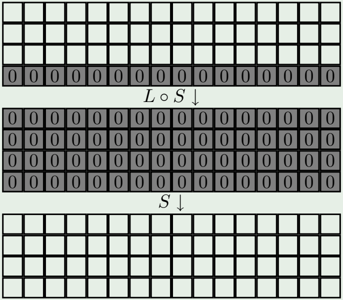


Bit-Pattern Based Integral Attack on PRESENT

Rounds 3 and 4

- Active S-boxes
  - Input: 1/2 distinct values repeated even number of times
  - Output: 1/2 distinct values repeated even number of times
- Linear
- S-box destroys balancedness

Distinguisher

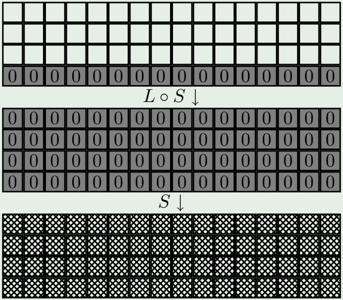


Bit-Pattern Based Integral Attack on PRESENT

Rounds 3 and 4

- Active S-boxes
  - Input: 1/2 distinct values repeated even number of times
  - Output: 1/2 distinct values repeated even number of times
- Linear
- S-box destroys balancedness

Distinguisher



4-round Key Recovery

- Guesses: 4 bits of K_4
- Initialize array $A[]$ of size 2^4
- Guess 4-bit subkey bits v of K_4
- Partially decrypt ciphertexts
- If Equation (1) does not hold, set $A[v] = 0$
- If only one entry such that $A[v] = 1$ is left, v is correct subkey bits
- Repeat for other 15 S-boxes
- Complexities
  - Data: $2 \times 2^4 = 2^5$
  - Time: $2 \times 2^4 \times 16 \times 2^4 = 2^{13}$
  - Memory: small

Bit-Pattern Based Integral Attack on PRESENT

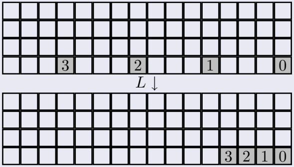
5-round Key Recovery

- Guesses
  - 4 bits of K_4
  - 16 bits K_5
- Initialize array A of size 2^{20}
- Guess 20-bit subkey bits v of K_4
- Partially decrypt ciphertexts
- If Equation (1) does not hold, set $A[v] = 0$
- If only one entry such that $A[v] = 1$ is left, v is correct subkey bits
- Complexities
  - Data: $5 \times 2^4 \approx 2^{6.4}$
  - Time: $(2^{20} + 2^{16} + \ldots + 1) \times 2^4 \times 3 + 2^{20} \approx 2^{25.7}$
  - Memory: small

Bit-Pattern Based Integral Attack on PRESENT

6-round Key Recovery

- Adding 1 round at the beginning



- Complexities
  - Data: $2^{4\times4} \times 2^{6.4} \approx 2^{22.4}$
  - Time: $2^{4\times4} \times 2^{25.7} \approx 2^{41.7}$
  - Memory: small

7-round Key Recovery

- Guesses
  - 4 bits of K_5
  - 16 bits K_6
  - 64 bits of K_7
- Complexities
  - Data: $20 \times 2^{16} \times 2^{4} \approx 2^{24.3}$
  - Time: $(2^{80} + 2^{76} + \ldots + 1) \times 2^{4} \times 2^{16} \approx 2^{100.1}$
  - Memory: 2^{77} bytes

Summary of Attacks on PRESENT

Rounds	Complexity		
	Data	Time	Memory
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

Analysis by designers

- Differential: 25-round characteristic (probability 2^{-100})
- Linear: 28-round linear approximation (bias 2^{-43})

The best 5-round differential attack on PRESENT requires on the order of 2^{20} CP. Our 5-round attack requires about 80 CP

CP = chosen plaintexts

Outline

1. Introduction
2. Bit-Pattern Based Integral Attack
   - Background
   - Type of Block Ciphers
   - Attack Algorithm
3. Application to Block Ciphers
   - PRESENT
   - Noekeon
   - Serpent
4. Discussion and Conclusion

Summary of Attacks on Noekeon

Rounds	Complexity		
	Data	Time	Memory
4	2^{17} CP	2^{26}	small
5	$2^{20.6}$ CP	$2^{108.1}$	2^{89} bytes

Related-key attack [Knudsen and Raddum, 2001]:

- 768 related keys with probability 2^{-32} for 16-round Noekeon

Our 3.5-round distinguisher for Noekeon with probability 1 is better than the 4-round differential trail with probability 2^{-48} predicted by the designers

Outline

1. Introduction
2. Bit-Pattern Based Integral Attack
   - Background
   - Type of Block Ciphers
   - Attack Algorithm
3. Application to Block Ciphers
   - PRESENT
   - Noekeon
   - Serpent
4. Discussion and Conclusion

Summary of Attacks on Serpent

Rounds	Attack	Complexity		
		Data	Time	Memory
4	Integral	2^{11} CP	2^{20}	small
5	Integral	$2^{13.6}$ CP	$2^{58.7}$	2^{44}
6	Integral	$2^{65.2}$ CP	$2^{110.7}$	2^{44}
	Differential [Kohno et al., 2000]	2^{83} CP	2^{90}	2^{44}
10	Linear [Collard et al., 2007]	2^{120} KP	2^{64}	2^{32}

The best differential attack on 5-round Serpent requires on the order of 2^{42} CP. Our 5-round attack requires $2^{13.6}$ CP

Discussion and Conclusion

Advantages

- Attack applies to bit-oriented block ciphers – order of texts plays important part
- Bit-pattern based integral attacks on analyzed ciphers are comparable to differential cryptanalysis over a few rounds – less chosen plaintexts

Limitations

- Differential cryptanalysis can be extended to more rounds – integral cryptanalysis can not be extended beyond a certain point
- Time complexity increases considerably as the number of cipher round increases

Discussion and Conclusion

Advantages

- Attack applies to bit-oriented block ciphers – order of texts plays important part
- Bit-pattern based integral attacks on analyzed ciphers are comparable to differential cryptanalysis over a few rounds – less chosen plaintexts

Limitations

- Differential cryptanalysis can be extended to more rounds – integral cryptanalysis can not be extended beyond a certain point
- Time complexity increases considerably as the number of cipher round increases

End

Thank you
Questions?

Anderson, R., Biham, E., and Knudsen, L. (1998).
Serpent: A Proposal for the Advanced Encryption Standard.
NIST AES Proposal.
Available at
http://www.cl.cam.ac.uk/~rja14/serpent.html.

Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsoe, C. (2007).
PRESENT: An Ultra-Lightweight Block Cipher.
In Paillier, P. and Verbauwhede, I., editors, *Cryptographic Hardware and Embedded Systems – CHES 2007, 9th International Workshop*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer-Verlag.

Collard, B., Standaert, F.-X., and Quisquater, J.-J. (2007).
Improved and Multiple Linear Cryptanalysis of Reduced Round Serpent.
In *Information Security and Cryptology, Third SKLOIS Conference, Inscrypt 2007, to appear*.

Daemen, J., Peeters, M., Assche, G. V., and Rijmen, V. (2000).
Nessie Proposal: NOEKEON.
First Open NESSIE Workshop.
Available at `http://gro.noekeon.org/`.

Knudsen, L. and Raddum, H. (2001).
On Noekeon.
NESSIE Phase 1 public reports.
Available at `https://www.cosic.esat.kuleuven.be/nessie/reports/`.

Kohno, T., Kelsey, J., and Schneier, B. (2000).
Preliminary Cryptanalysis of Reduced-Round Serpent.
In *The Third Advanced Encryption Standard Candidate Conference*, pages 195–211. NIST.
Available at `http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/aes3conf.htm`.