Baksheesh

Mohit Thakre, Nishchay Rajput and Prajapati Harsh Pareshkumar

Indian Institute of Technology, Bhilai

Abstract.
Our paper presents the **BAKSHEESH**, which is a lightweight block cipher designed over the **GIFT-128**. It focuses over the efficiency without compromising security against classical cryptanalytic methods from the GIFT-128. It has comparatively less rounds i.e. 35 as compared to the GIFT-128(40 rounds). It is smaller in design with comparable resistance to differential and linear attacks. It has unique 4-bit S-box with a single not-trivial Linear Structure(LS), which balances the simplicity and security. Design for this cipher has improved hardware efficiency and includes the less number of gate and better resistance to side-channel attacks. In the implementation of BAKSHEESH, it uses the **full-round key XOR** compared to the half-round key XOR in GIFT-128. BAKSHEESH sets a new standard for lightweight cipher design, offering a viable alternative to its predecessors.

Contents

1 Introduction

The rapid proliferation of Internet of Things (IoT) devices has underscored the importance of lightweight cryptography, where constrained hardware environments demand highly efficient and secure encryption mechanisms. Lightweight cryptographic solutions aim to minimize computational and hardware costs while maintaining robust security against classical cryptanalytic techniques. This field has gained significant traction, as evidenced by recent cipher designs like GIFT and PYJAMASK-128, alongside global competitions such as CAESAR and NIST's Lightweight Cryptography Initiative. Building upon this trend, the BAKSHEESH cipher emerges as a successor to the widely acclaimed lightweight cipher, GIFT-128. While GIFT-128 set benchmarks in terms of efficiency and security, advancements in cryptographic research over the past few years have paved the way for even greater optimization. BAKSHEESH takes this evolution further by introducing a novel 4-bit S-Box with a non-trivial Linear Structure (LS), an innovation that enables a delicate balance between simplicity, efficiency, and security. The BAKSHEESH cipher retains the core design philosophy of GIFT-128, incorporating the same bit-permutation layer to ensure backward compatibility and ease of implementation. However, it achieves a 12.5% reduction in rounds, with 35 rounds compared to GIFT-128's 40, leading to a smaller overall design. The S-Box in BAKSHEESH boasts key properties such as a Linear Branch Number (LBN) of 3 (the theoretical upper bound for 4×4 S-Boxes), an algebraic degree of 2, and a low gate count of only 3 AND operations. These properties make BAKSHEESH exceptionally efficient for both hardware and software implementations, while maintaining resistance to classical attacks like differential and linear cryptanalysis. This paper explores the BAKSHEESH cipher in detail, examining its design, security, and performance. Key contributions include the analysis of its S-Box properties, implementation efficiency, and cryptanalytic resistance. The goal is to demonstrate how unconventional design choices, such as the use of a non-trivial LS S-Box, can lead to significant advancements in lightweight cryptographic designs.

2 Construction

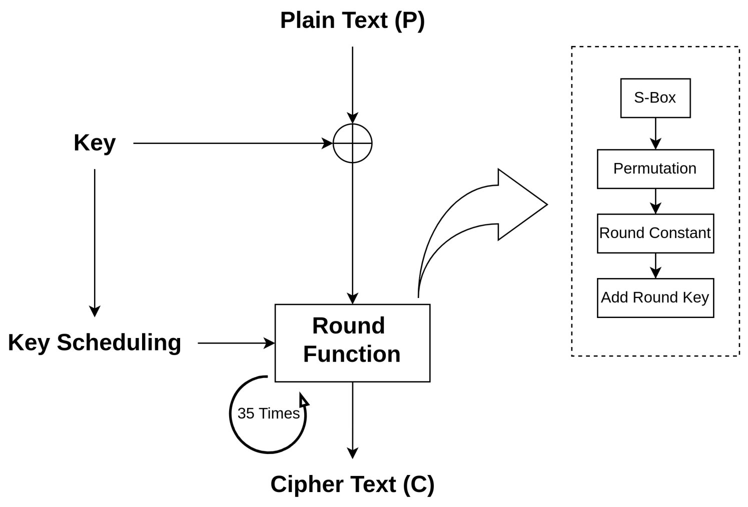


Figure 1: Baksheesh Construction

2.1 SubCells (S-Box)

Table 1: S-Box for BAKSHEESH

Input	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Output	3	0	6	d	b	5	8	e	c	f	9	2	4	a	7	1

The S-Box take an 4-bit input and correspondingly gives and 4-bit output as shown in **Table 1**. The SBox used in Baksheesh is $S = 306\text{DB}58\text{ECF}924\text{A}71$. This SBox exhibits several desirable cryptographic properties. It has a single non-zero linear structure (LS) at position 8, a linear branch number (LBN) of 3, and a differential branch number (DBN) of 2. **Table 2** compares the cryptographic properties of lightweight SBoxes, where 'AD' denotes algebraic degree and 'Nl' refers to non-linearity.

Table 2: Cryptographic properties of few lightweight cipher

		DBN	LBN	DU	AD (max)	AD (min)
Baksheesh	306DB58ECF924A71	2	3	16	2	2
PRESENT	C56B90AD3EF84712	3	2	4	3	2
SKINNY-64	C6901A2B385D4E7F	2	2	4	3	2
GIFT	1A4C6F392DB7508E	2	2	6	3	2
PYJAMASK-128	2D397BA6E0F4851C	2	2	4	3	2
PRINCE	BF32AC916780E5D4	2	2	4	3	3
KLEIN	74A91FB0C3268ED5	2	2	4	3	3
LED	C56B90AD3EF84712	3	2	4	3	2
PUFFIN	D7329AC1F45E60B8	2	2	4	3	3
PRINT(3-bit s-box)	01367452	2	2	2	2	2
Rectangle	65CA1E79B03D8F42	2	2	4	3	2
SQUARE	AES-SBox	2	2	4	7	-
MIDORI	1053E2F7DA9BC846	2	2	4	3	2
	1053E2F7DA9BC846	3	2	4	3	2

;

The coordinate functions of the Baksheesh SBox, represented in their algebraic normal form (ANF), are given as:

$$\begin{aligned}
y_0 &= x_0x_2 \oplus x_0 \oplus x_1 \oplus x_3 \oplus 1,\\
y_1 &= x_0 \oplus x_1x_2 \oplus x_3 \oplus 1,\\
y_2 &= x_0x_2 \oplus x_1x_2 \oplus x_1 \oplus x_3,\\
y_3 &= x_0x_1 \oplus x_0x_2 \oplus x_2 \oplus x_3.
\end{aligned}$$

Implementation Costs The Baksheesh SBox is lightweight, as demonstrated by its ASIC benchmarks shown in **Table 3**. The benchmarks, obtained using a look-up-based implementation approach, show that this SBox is more cost-efficient than most recent 4×4 SBoxes. However, it is slightly less efficient than the SKINNY-64 SBox when implemented using the Faraday 65nm library.

Table 3: AISC Benchmark for lightweight S-Box

		Cost (Gate Equivalent)		
		UMC 65nm	Faraday 65nm	STM 130nm
GIFT	1A4C6F392DB7508E	28	22	21
PYJAMASK-128	2D397BA6E0F4851C	28	26	22
SKINNY-64	C6901A2B385D4E7F	21	16	21
Baksheesh	306DB58ECF924A71	21	19	21

Branch Number Properties The linear branch number (LBN) of the Baksheesh SBox is 3, which is the theoretical upper limit for any 4×4 SBox. Any 4×4 SBox with LBN 3 must have at least one non-zero LS. While the Baksheesh SBox achieves this optimal LBN with a minimal LS count, our search for SBoxes with a differential branch number (DBN) of 3 showed that such SBoxes typically have at least three non-zero LS values. For example, the SBox $S =$ 126CDE39F58BA047 has DBN 3 but includes three non-zero LS values.

SBoxes with multiple non-zero LS values tend to slow down the propagation of differential and linear trails, which may necessitate more encryption rounds for adequate security. In contrast, the Baksheesh SBox, with its 1LS/3LBN/2DBN, strikes a balance between efficient implementation and cryptographic strength. This balance helps to reduce the number of rounds while keeping the AND gate count low, contributing to the overall efficiency of the cipher.

2.2 PermBits (Bit Permutations as Linear Layer)

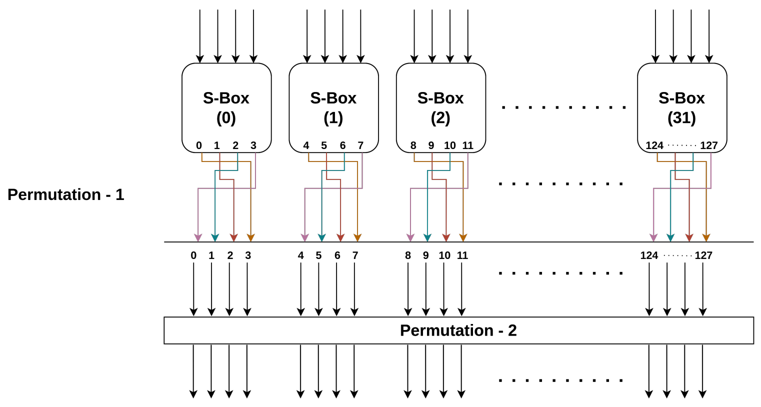


Figure 2: Permutation Layer

Here we can see in **Figure 2** that we have two permutation layer i.e **Permutation Layer 1** and **Permutation Layer 2** which each has relation corresponding with the input bit to the output bit.

$$\textbf{Permutation 1}: i^{th}bit \leftarrow (3 - (\text{i mod } 4)) + (4 * \left\lfloor \frac{i}{4} \right\rfloor) \text{ where, i } \in \{0, 1, 2...127\}.$$

$$\textbf{Permutation 2}: i^{th}bit \leftarrow ((\text{i mod } 4) + (\left\lfloor \frac{i}{16} \right\rfloor \text{ x } 4)) + 32 * ((\text{i - }((\left\lfloor \frac{i}{4} \right\rfloor) \text{ mod } 4)\text{) mod } 4) \text{ where, i } \in \{0, 1, 2...1$$

Table 4: Permutation-2 $P128$ Mapping

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P128(i)$	0	33	66	99	96	1	34	67	64	97	2	35	32	65	98	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P128(i)$	4	37	70	103	100	5	38	71	68	101	6	39	36	69	102	7
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P128(i)$	8	41	74	107	104	9	42	75	72	105	10	43	40	73	106	11
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P128(i)$	12	45	78	111	108	13	46	79	76	109	14	47	44	77	110	15
i	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
$P128(i)$	16	49	82	115	112	17	50	83	80	113	18	51	48	81	114	19
i	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
$P128(i)$	20	53	86	119	116	21	54	87	84	117	22	55	52	85	118	23
i	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
$P128(i)$	24	57	90	123	120	25	58	91	88	121	26	59	56	89	122	27
i	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
$P128(i)$	28	61	94	127	124	29	62	95	92	125	30	63	60	93	126	31

Permutation layer operates over the 128-bit state. It is a **bit permutation** which shuffles the bit position as defined in **Table 2**.

$$b_{\mathrm{P128}(i)} \leftarrow b_i, \quad \forall i \in \{0, 1, \ldots, 127\},$$

2.3 AddConstants (Round Constants XOR Layer)

Table 5: Round Constants at Tap Positions

Round	Tap 8	Tap 13	Tap 19	Tap 35	Tap 67	Tap 106
1	2	33	16	9	36	19
2	40	53	26	13	38	51
3	56	61	62	31	14	7
4	34	49	24	45	54	59
5	28	47	22	43	20	11
6	4	3	32	17	8	–

Note: The bit at the last tap position (106) toggles in each round.

In the **AddConstant layer** the certain bits of the internal state are **XORed at specific position** also known as **tap position** which are defined in **Table 3**. These position are fixed and chosen such that it provides the maximum diffusion across the rounds. Not all bits are XORed as to reduce the computational complexity and to ensure that the cipher is lightweight.

2.4 Key Schedule Routine and AddRoundKey Layer

The initial key schedule of BAKSHEESH was designed to be simple, where the master key is directly XORed with the state, to minimize implementation overhead. However, this approach was found vulnerable to an invariant **subspace attack**, primarily due to the sparse round constants, as only 6 fixed-bit positions were modified during the constant addition operation. To counter this weakness, the key schedule was adjusted to include a 1-bit right rotation for each round, represented as:

$$k_{j+1} \leftarrow k_j \gg 1.$$

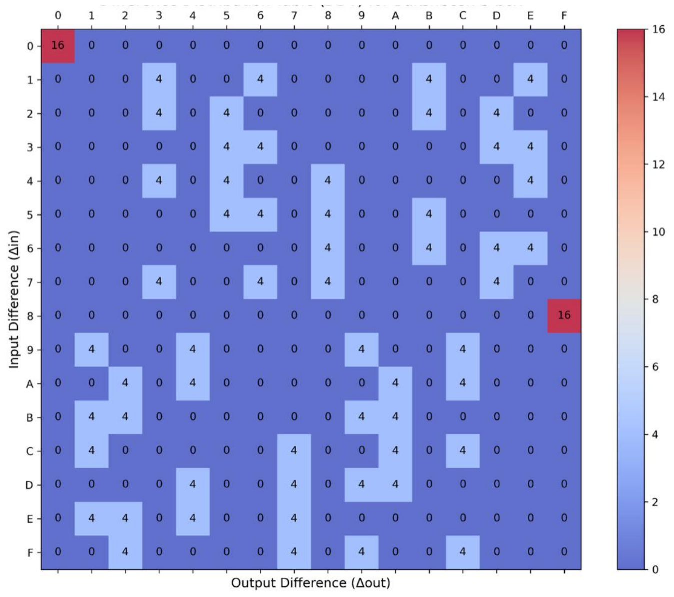
Following this modification, extensive analysis revealed no signs of invariant subspace attacks. Additionally, the full 128-bit key is XORed with the state, replacing the half-key XOR used in GIFT-128. While this adjustment increases the cost, it provides a more robust and secure design.

3 Differential and Linear Analysis

An S-box (Substitution box) is a fundamental nonlinear component in block ciphers, providing confusion in the encryption process. The effectiveness of an S-box is measured by analyzing its resistance to cryptanalysis techniques such as Differential Cryptanalysis and Linear Cryptanalysis. Results of the analysis of S-Box from the DDT and LAT are explained below.

3.1 DDT analysis

The Difference Distribution Table (DDT) quantifies how input differences (ΔX) map to output differences (ΔY) under the S-box transformation. It is crucial for the assessment of resistance to Differential Cryptanalysis.


Input Difference (Δin)
Output Difference (Δout)
16
14
12
10
8
6
4
2
0

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	0	4	0	0	0	0	4	0	0	4	0
2	0	0	0	4	0	4	0	0	0	0	0	4	0	4	0	0
3	0	0	0	0	0	4	4	0	0	0	0	0	0	4	4	0
4	0	0	0	4	0	4	0	0	4	0	0	0	0	0	4	0
5	0	0	0	0	0	4	4	0	4	0	0	4	0	0	0	0
6	0	0	0	0	0	0	0	0	4	0	0	4	0	4	4	0
7	0	0	0	4	0	0	4	0	4	0	0	0	0	4	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16
9	0	4	0	0	4	0	0	0	0	4	0	0	4	0	0	0
A	0	0	4	0	4	0	0	0	0	0	4	0	4	0	0	0
B	0	4	4	0	0	0	0	0	0	4	4	0	0	0	0	0
C	0	4	0	0	0	0	0	4	0	0	4	0	4	0	0	0
D	0	0	0	0	4	0	0	4	0	4	4	0	0	0	0	0
E	0	4	4	0	4	0	0	4	0	0	0	0	0	0	0	0
F	0	0	4	0	0	0	0	4	0	4	0	0	4	0	0	0

Figure 3: Difference Distribution Table

The table shows:

- Row indices correspond to all possible input differences (0-16 for 4-bit S-Box).
- Column indices correspond to all possible output differences (0-16 for 4-bit S-Box).
- Each entry represents the number of input-output pairs that satisfy the equation:

$$S(X) \oplus S(X \oplus \Delta X) = \Delta Y$$

- For this S-box, the following key observations are made:
- Differential Uniformity: The highest non-zero entry in the DDT is 16, meaning that for some input differences (ΔX except 0), every output difference (ΔY except 0) is equally likely. This indicates that the S-box has 16-uniformity, which is significantly higher than the ideal differential uniformity of 2 seen in robust S-boxes like AES. A differential uniformity of 16 is weak and makes the cipher vulnerable to Differential Cryptanalysis.

3.2 Differential Branch Number/weight

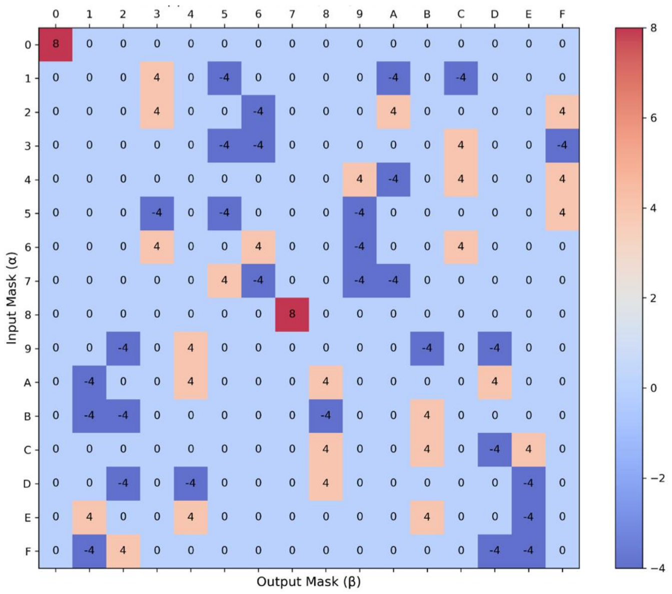
The branch number is another key property that measures the diffusion capability of the S-box. It is calculated as:

where the Hamming weight is the number of non-zero bits in x and S(x). For this S-box:

- Branch Number: 2.
- This is a relatively low value, indicating limited diffusion. A higher branch number (closer to the size of the S-box, i.e., 4) is desirable for strong mixing of input and output bits.
- In Baksheesh Cipher, S-BOX is chosen considering the good balance in cost for implementation and number of rounds as 1LS/3LBN/2DBN rather than choosing an S-BOX with higher DBN.

3.3 Linear Approximation Table

The Linear Approximation Table (LAT) measures correlations between linear combinations of input and output bits. Each entry is calculated as:



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	-4	0	0	0	0	-4	0	-4	0	0	0
2	0	0	0	4	0	0	-4	0	0	0	4	0	0	0	0	4
3	0	0	0	0	0	-4	-4	0	0	0	0	0	4	0	0	-4
4	0	0	0	0	0	0	0	0	0	4	-4	0	4	0	0	4
5	0	0	0	-4	0	-4	0	0	0	-4	0	0	0	0	0	4
6	0	0	0	4	0	0	4	0	0	-4	0	0	4	0	0	0
7	0	0	0	0	0	4	-4	0	0	-4	-4	0	0	0	0	0
8	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0
9	0	0	-4	0	4	0	0	0	0	0	0	-4	0	-4	0	0
A	0	-4	0	0	4	0	0	0	4	0	0	0	0	4	0	0
B	0	-4	-4	0	0	0	0	0	-4	0	0	4	0	0	0	0
C	0	0	0	0	0	0	0	0	4	0	0	4	0	-4	4	0
D	0	0	-4	0	-4	0	0	0	4	0	0	0	0	0	-4	0
E	0	4	0	0	4	0	0	0	0	0	0	4	0	0	-4	0
F	0	-4	4	0	0	0	0	0	0	0	0	0	0	-4	-4	0

Figure 4: Linear Approximation Table

where a and b are the input and output masks, respectively. The LAT provides insights into the S-box's resistance to Linear Cryptanalysis. Key observations from the LAT:

- The **maximum absolute value** in the LAT is 8 (bias), indicating the strongest correlation between linear combinations of input and output bits. This value indicates that for input mask 8 and output mask 7, you will definitely be able to find one bit of the key from the linear cryptanalysis.
- High correlation values suggest that the S-box is susceptible to **Linear Cryptanalysis**, as attackers can exploit these correlations to predict output values with greater probability.

3.4 Comparative Analysis

The following table compares the properties of this S-box with S-boxes from other well-known ciphers:

Table 6: Comparative Analysis

Cipher	Differential Uniformity	Branch Number	Maximum Linear Bits	Size (in bits)	Resistance to Attacks
Baksheesh	16	2	8	4x4	Weak resistance to differential and linear attacks.
AES	2	5	4	8x8	Strong resistance to cryptanalysis.
PRESENT	4	3	6	4x4	Good balance for lightweight security.

3.5 Practical Implication

The cryptographic properties of this S-box indicate the following:

- Weak Resistance to Differential Cryptanalysis: The high differential uniformity (16) implies that differential attacks are feasible, as attackers can predict output differences for given input differences with high certainty.
- Limited Diffusion: The low branch number (2) suggests that changes in input bits are not well-diffused into the output, reducing the effectiveness of confusion and diffusion layers in the cipher.
- Moderate Linear Correlation: The LAT shows significant correlations (8), making the S-box moderately susceptible to linear cryptanalysis. Attackers can exploit these correlations to predict key bits more efficiently.

4 Cryptanalysis

4.1 Differential Cryptanalysis

For the differential cryptanalysis, **SAT models** are used to find the optimal trails. MILP-based autmoation approach could not reach the number of round as high as the SAT. So we will be see the analysis based on SAT model.

4.1.1 Finding Differential Trails

In SAT model, we use **CNF language** to describe the constraints. Since DDT values are even(i.e. 4 and 16), so only one variable needed to represent it.

$$p = 1 \textbf{ for } DDT[i][j] = 4$$

$$p = 0 \text{ for } DDT[i][j] = 16$$

It would take 4 varibles for the input bits, 4 varibles for the output bit and 1 for the probability for an S-Box. Each S-Box has 27 CNF clauses.

Table 7: Optimal differential bounds for BAKSHEESH (single trail), where p denotes the probability.

Round	1	2	3	4	5	6	7	8	9	10	11
$-\log_2 p$	0	2	4	8	14	20	30	40	48	54	60
Round	12	13	14	15	16	17	18	19	20	21	22
$-\log_2 p$	68	76	84	92	100	110	120	126	132	140	148

For round 1, $-log(p) = 0$ means trail has probability of $2^0 = 1(100\%)$.As the number of rounds increases the, the probability of differential trail decreases, which indicates the differential cryptanalysis become harder for more rounds(due to avalanche effect, diffusion and non-linear transformation).

4.2 A 19-round Differential Key-recovery Attack

Considering the 18-round differential trail with a probability of 2^{-120}, as illustrated in **Table 7**, we present a key-recovery attack on the 19-round BAKSHEESH cipher by extending the distinguisher with one additional round. The input difference for this distinguisher is:

$$(0x0d00940040a00000, 0x0000000000000000),$$

and the output difference is:

$$(0x8800890044004c00, 0x2200260011001300).$$

The specific steps for key-recovery are as follows:

1. Choose 2^{121} pairs of plaintexts with differences equal to the input differences of the 19-round differential trail.
2. Filter out incorrect pairs using the inactive bits of the ciphertexts; there are 257 pairs remaining.
3. Initialize a list of 2^{64} empty counters to guess 64 bits of the subkey RK_{19}.
4. For all 257 pairs, perform a guess-and-filter procedure to identify candidate keys and update the corresponding counters.

The subkey with the maximum counter is selected as the correct subkey. The complexities of this process are:

- **Data Complexity**: 2^{122}
- **Memory Complexity**: 2^{64}
- **Time Complexity**: 2^{121}

To recover the remaining 64 bits of the 128-bit master key, a brute-force search with a time complexity of 2^{64} is performed. Thus, the total time complexity to recover the full 128-bit key is 2^{121}.

4.3 Integral Cryptanalysis on Round-reduced BAKSHEESH

In this section, we will first give an overview of our framework for mounting an $(r+2)$-round key-recovery attack using a given r-round integral distinguisher. Then we finally used it and find 7- and 8-round distinguishers to achieve 9- and 10-round practical attacks, and a 13-round distinguisher to do a theoretical 15-round key-recovery attack.

Main Idea: *Equivalent Integral Property*

4.3.1 Steps to do the Attack:

1. **Query**: For the input plaintext X_0, the output at the round $r+2$ is X^{r+3}.

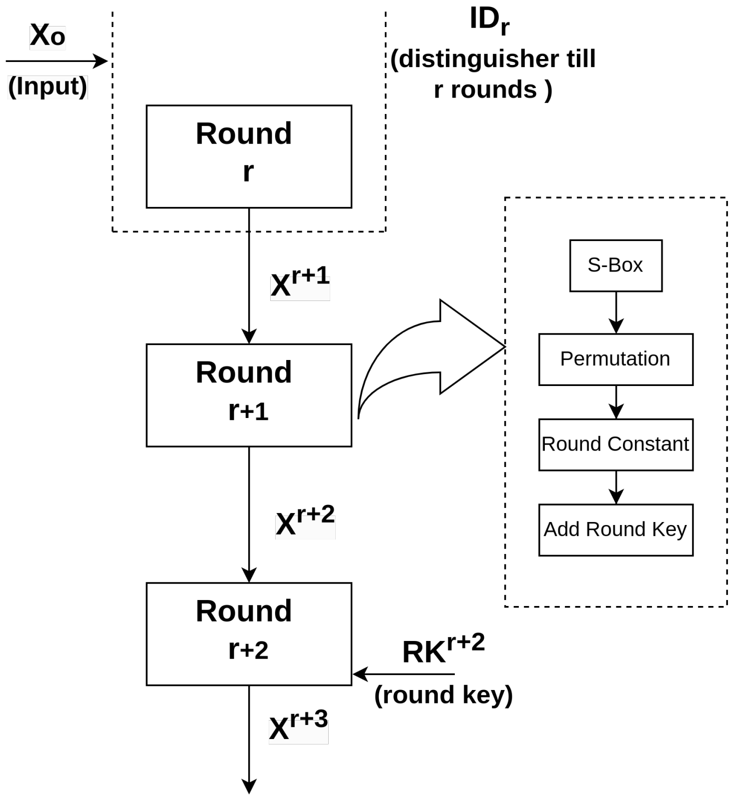


Figure 5: Flow of Integral Cryptoanalysis

2. **Guess:** Guess has to be made in the round key bit of round $r+2$ (RK^{r+2}). Firstly, the plaintexts for query are chosen as equal to 2^d. 'd' denotes the number of active bits found.

Table 8: The corresponding bit positions of RK^{r+2} that need to be guessed for detecting the integral property of X^{r+1} at the bit position $4i+3$ $(0 \leq i \leq 31)$.

X^{r+1}	Guessed key RK^{r+2}	Set	Constant Set
3	0, 33, 66, 99, 8, 41, 74, 107, 16, 49, 82, 115	K_0	K'_0
7	24, 57, 90, 123, 0, 33, 66, 99, 8, 41, 74, 107	K_1	
11	16, 49, 82, 115, 24, 57, 90, 123, 0, 33, 66, 99	K_2	K'_0
15	8, 41, 74, 107, 16, 49, 82, 115, 24, 57, 90, 123	K_3	
19	96, 1, 34, 67, 104, 9, 42, 75, 112, 17, 50, 83	K_4	K'_1
23	120, 25, 58, 91, 96, 1, 34, 67, 104, 9, 42, 75	K_5	
27	112, 17, 50, 83, 120, 25, 58, 91, 96, 1, 34, 67	K_6	K'_1
31	104, 9, 42, 75, 112, 17, 50, 83, 120, 25, 58, 91	K_7	
35	64, 97, 2, 35, 72, 105, 10, 43, 80, 113, 18, 51	K_8	K'_2
39	88, 121, 26, 59, 64, 97, 2, 35, 72, 105, 10, 43	K_9	
43	80, 113, 18, 51, 88, 121, 26, 59, 64, 97, 2, 35	K_{10}	K'_2
47	72, 105, 10, 43, 80, 113, 18, 51, 88, 121, 26, 59	K_{11}	
51	32, 65, 98, 3, 40, 73, 106, 11, 48, 81, 114, 19	K_{12}	K'_3
55	56, 89, 122, 27, 32, 65, 98, 3, 40, 73, 106, 11	K_{13}	
59	48, 81, 114, 19, 56, 89, 122, 27, 32, 65, 98, 3	K_{14}	K'_3
63	40, 73, 106, 11, 48, 81, 114, 19, 56, 89, 122, 27	K_{15}	
67	120, 25, 58, 91, 128, 33, 66, 99, 136, 41, 74, 107	K_{16}	K'_4
71	24, 57, 90, 123, 0, 33, 66, 99, 8, 41, 74, 107	K_{17}	
75	16, 49, 82, 115, 24, 57, 90, 123, 0, 33, 66, 99	K_{18}	K'_4
79	8, 41, 74, 107, 16, 49, 82, 115, 24, 57, 90, 123	K_{19}	
83	96, 1, 34, 67, 104, 9, 42, 75, 112, 17, 50, 83	K_{20}	K'_5
87	120, 25, 58, 91, 96, 1, 34, 67, 104, 9, 42, 75	K_{21}	
91	112, 17, 50, 83, 120, 25, 58, 91, 96, 1, 34, 67	K_{22}	K'_5
95	104, 9, 42, 75, 112, 17, 50, 83, 120, 25, 58, 91	K_{23}	
99	64, 97, 2, 35, 72, 105, 10, 43, 80, 113, 18, 51	K_{24}	K'_6
103	88, 121, 26, 59, 64, 97, 2, 35, 72, 105, 10, 43	K_{25}	
107	80, 113, 18, 51, 88, 121, 26, 59, 64, 97, 2, 35	K_{26}	K'_6
111	72, 105, 10, 43, 80, 113, 18, 51, 88, 121, 26, 59	K_{27}	
115	32, 65, 98, 3, 40, 73, 106, 11, 48, 81, 114, 19	K_{28}	K'_7
119	56, 89, 122, 27, 32, 65, 98, 3, 40, 73, 106, 11	K_{29}	
123	48, 81, 114, 19, 56, 89, 122, 27, 32, 65, 98, 3	K_{30}	K'_7
127	40, 73, 106, 11, 48, 81, 114, 19, 56, 89, 122, 27	K_{31}	

Active bits: The bits in X_0 are those which are together as a group varied with 2^d values and other remaining bits are known as the **unknown bits**, which are kept constant.

3) Approach of selecting active bits: For selecting 'd' active bits, consecutive 'd' bits are chosen and tested to see if they give maximum number of balanced bits. For this 'd' active bits are chosen and provided with 2^d different values. The calculated results are as follows:

Table 9: The Active bits found by the given approach.

Dist.	Active bits	Number
ID_7	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	14
ID_8	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31	30

ID_r: Distinguisher till r-rounds.

For ID_7, $d = 14$
For ID_8, $d = 30$

4) Filtering (Final Step): As we know,

$$\bigoplus_{X^0 \in \mathbb{D}} x^r_{4i+3} = \bigoplus_{X^0 \in \mathbb{D}} \left(x^{r+1}_{P(4i)} \oplus x^{r+1}_{P(4i+1)} \oplus x^{r+1}_{P(4i+2)} \right)$$

For the value of X^{r+2},

If R.H.S $= 0$, then $x^r_{4i+3} = 0$ $\Rightarrow$ Select the Key bits guess.
If R.H.S $\neq 0$, then $x^r_{4i+3} \neq 0$ $\Rightarrow$ Eliminate the Key bits guess.

5) Repeat Filtering for 3 times and then an exhaustive search:

Repeat step 4th 3 times with different 2^d plaintexts, and thus for all $0 \leq s \leq 7$, there are in total 2^{16} guessed Key bits and every 3 times 2^4 Keys are eliminated (assuming $b_0 = 4$ of the s^{th} group).

Thus we are left with 2^4 key guesses, those are checked with exhaustive key search.

Proof of the Attack :

Proposition :

$$\bigoplus_{X^0 \in \mathbb{D}} x^r_{4i+3} = \bigoplus_{X^0 \in \mathbb{D}} \left(x^{r+1}_{P(4i)} \oplus x^{r+1}_{P(4i+1)} \oplus x^{r+1}_{P(4i+2)} \right)$$

Proof :

We know that,

$$x^r_{4i+3} = y^r_{4i+2} \oplus y^r_{4i+1} \oplus y^r_{4i} \quad (1)$$

'i' denotes the i-th S-box (S). The equation (1) is followed by the following property 1 of the S-box in BAKSHEESH Cipher.

Property 1 :

For BAKSHEESH's Sbox, there is a linear trail $1000 \rightarrow 0111$ with correlation 1.

Property 2 :

For the inverse of BAKSHEESH's Sbox, there is a differential transition $1111 \rightarrow 1000$ with probability 1.

$$x^{r+1}_{P(j)} = y^r_j \oplus rk^{r+1}_{P(j)} \oplus rc^r_{P(j)}, \quad 0 \leq j \leq 127,$$

where $rc^r_{P(j)} = c^r_{P(j)}$ if $P(j) \in \{8, 13, 19, 35, 64, 106\}$, otherwise $rc^r_{P(j)} = 0$. Therefore, the bit x^r_{4i+3} can be represented as

$$x^r_{4i+3} = x^{r+1}_{P(4i)} \oplus x^{r+1}_{P(4i+1)} \oplus x^{r+1}_{P(4i+2)} \oplus ck^r_i, \tag{1}$$

where $ck^r_i = rk^{r+1}_{P(4i)} \oplus rk^{r+1}_{P(4i+1)} \oplus rk^{r+1}_{P(4i+2)} \oplus rc^{r+1}_{P(4i)} \oplus rc^{r+1}_{P(4i+1)} \oplus rc^{r+1}_{P(4i+2)}$. Note that ck^r_i is a constant determined by the subkey and round constant, since we have 2^d plaintexts, of $(r+1)$-th round, which gives that

$$ck^r_i = 0. \tag{2}$$

Thus, we have

$$x^r_{4i+3} = (x^{r+1}_{P(4i)} \oplus x^{r+1}_{P(4i+1)} \oplus x^{r+1}_{P(4i+2)} \oplus ck^r_i) = (x^{r+1}_{P(4i)} \oplus x^{r+1}_{P(4i+1)} \oplus x^{r+1}_{P(4i+2)}). \tag{3}$$

Exploring related key bits (Introduction of K'_s). For a given r-round distinguisher, let's assume that the position of bit $4i+3$ $(0 \leq i \leq 31)$, i.e., x^{r+1}_{4i+3}, exhibits the balanced integral property. We can filter the guessed values of partial bits of the subkey RK^{r+2} associated with x^{r+1}_{4i+3} by checking the balanced property of $x^{r+2}_{P(4i)} \oplus x^{r+2}_{P(4i+1)} \oplus x^{r+2}_{P(4i+2)}$, whose value can be determined using the known $(r+2)$-th round output, i.e., X^{r+3}, and the guessed key bit rk^{r+2}.

Specifically, for the instance of $x^{r+2}_{P(4i+j)}$ $(j \in \{0, 1, 2\})$, we first need to calculate the 4 bits $(y^{r+2}_{4\ell_j+3}, y^{r+2}_{4\ell_j+2}, y^{r+2}_{4\ell_j+1}, y^{r+2}_{4\ell_j})$ as follows:

$$y^{r+2}_{4\ell_j+m} = x^{r+3}_{P(4\ell_j+m)} \oplus rk^{r+2}_{P(4\ell_j+m)}, \quad 0 \leq m \leq 3 \tag{4}$$

according to the round function, where $\ell_j = \left\lfloor \frac{P(4i+j)}{4} \right\rfloor$ and $rk^{r+2}_{P(4\ell_j+m)}$ is a guessed key bit. By inverting the S-box $S^{ei}_{\ell_j}$, the value of $x^{r+2}_{P(4i+j)}$ is obtained. Consequently, there are a total of 12 key bits that need to be guessed to detect the balanced property of x^{r+1}_{4i+3}. We list the key bits of RK^{r+2} corresponding to each x^{r+1}_{4i+3} $(0 \leq i \leq 31)$ in Table 7. Denote K_i $(0 \leq i \leq 31)$ as the set containing the bit positions of RK^{r+2} corresponding to x^{r+1}_{4i+3}, we observe the following from Table 7.

Observation 1. The union of any b $(b \geq 2)$ sets among the s-th $(0 \leq s \leq 7)$ set tuple $(K_{4s}, K_{4s+1}, K_{4s+2}, K_{4s+3})$ is a constant set denoted by K^b_s that contains 16 bit positions.

The data is with the reference to Table 7.

For example, regarding (K_0, K_1, K_2, K_3), we have

$$\begin{aligned} K'_0 &= K_0 \cup K_1 = K_0 \cup K_2 = K_0 \cup K_3 = K_1 \cup K_2 = K_1 \cup K_3 = K_2 \cup K_3 \\ &= K_0 \cup K_1 \cup K_2 = K_0 \cup K_1 \cup K_3 = K_1 \cup K_2 \cup K_3 \\ &= K_0 \cup K_1 \cup K_2 \cup K_3 \\ &= \{0, 33, 66, 99, 8, 41, 74, 107, 16, 49, 82, 115, 24, 57, 90, 123\}. \end{aligned}$$

Thus, if there are b_0 $(b_0 \geq 2)$ balanced bits among $(x^{r+1}_3, x^{r+1}_7, x^{r+1}_{11}, x^{r+1}_{15})$, we can use these balanced bits to filter the guessed key bits, whose positions are within K'_0. Specifically, the filtering strength is 2^{-b_0}, meaning a wrongly guessed key can pass one filtering with a probability of 2^{-b_0}. Here, we define detecting the balanced property for all remaining guessed-key candidates as one filtering.

For instance, assuming all 4 bits of $(x_3^{r+1}, x_7^{r+1}, x_{11}^{r+1}, x_{15}^{r+1})$ are balanced. Initially, there are 2^{16} guessed values for the 16 bits of subkey RK^{r+2} as shown in K'_0. For each guessed value, we decrypt 2^d X^{r+3} generated by the chosen X^0 to obtain the information of $(x_3^{r+1}, x_7^{r+1}, x_{11}^{r+1}, x_{15}^{r+1})$, and compute the XOR sum for these 4 bits. If the XOR sum for the 4 bits is all 0s, we retain this guessed value as a candidate; otherwise, we discard it. After 2^{16} instances (termed one filtering), there are theoretically 2^{12} remaining candidates. Repeating this process once more, i.e., filtering a second time, results in 2^8 candidates remaining.

Filtering Analysis: For an r-round distinguisher, let b_s denote the number of balanced bits within the s-th bit group $(x_{16s+3}^{r+1}, x_{16s+7}^{r+1}, x_{16s+11}^{r+1}, x_{16s+15}^{r+1})$ where $0 \leq s \leq 7$. Let C_s represent the set containing the candidates of the 16 key bits whose positions are in K'_s for $0 \leq s \leq 7$. Initially, C_s comprises 2^{16} candidates before any filtering occurs. In theory, after t_s rounds of filtering C_s, $2^{16-t_s \cdot b_s}$ candidates will remain in C_s. Specifically, when $t_s \cdot b_s \geq 16$, the correct value can be identified. However, this theoretical prediction may not align with real-world outcomes, as highlighted in the following observation.

Observation 2: For each $0 \leq s \leq 7$, there are always 2^4 surviving candidates (including the correct value) in C_s, no matter how many times we filter them. In order to illustrate this observation more intuitively, we use the case of $s = 0$ as an example. The related bits of state and subkey for detecting the integral property of $(x_{15}^{r+1}, x_{11}^{r+1}, x_7^{r+1}, x_3^{r+1})$ are simply depicted in Figure 2. Coincidentally, the 12 related bits of X^{r+2} are distributed as the 3 LSBs of 4 Sboxes. Assuming all bits of $(x_{15}^{r+1}, x_{11}^{r+1}, x_7^{r+1}, x_3^{r+1})$ are balanced, then the filtering strength is 2^{-4} and 2^4 candidates will survive in C_0 after 3 filterings. Let us focus on S_0. From Property 2, we know that the value of $(x_2^{r+2}, x_1^{r+2}, x_0^{r+2})$ will not be changed if we simultaneously turn over the 4 bits $(y_3^{r+2}, y_2^{r+2}, y_1^{r+2}, y_0^{r+2})$. Note that

$$\begin{aligned}
y_3^{r+2} &= rk_{99}^{r+3} \oplus x_{99}^{r+3}, \\
y_2^{r+2} &= rk_{66}^{r+2} \oplus x_{66}^{r+2}, \\
y_1^{r+2} &= rk_{33}^{r+3} \oplus x_{33}^{r+3}, \\
y_0^{r+2} &= rk_0^{r+2} \oplus x_0^{r+3}.
\end{aligned}$$

In other words, if the correct values of $(rk_{99}^{r+2}, rk_{66}^{r+2}, rk_{33}^{r+2}, rk_0^{r+2})$ are (v_3, v_2, v_1, v_0), then the guessed values $(v_3 \oplus 1, v_2 \oplus 1, v_1 \oplus 1, v_0 \oplus 1)$ will always pass the filtering process. Essentially, there are two 4-bit key values that can pass the filtering for each of the 4 S-boxes: one being the correct value and the other being its negation. Thus, there are a total of 2^4 candidates that will remain in C_0.

Observation 2 tells us that we can recover at most 96 bits of information of the 128-bit subkey using integral distinguishers. Then, the remaining 32-bit information can be determined by exhaustive searching.

4.4 Attacks for Key-recovery

In this sectionn, we will utilize the made distinguishers(ID_7 and ID_8) to do two practical and a theoretical key-recovery attacks based on the framework of baksheesh cipher. Firstly, defining some notations. For an r-round distinguisher, we denote b_s the number of balanced bits among the s-th bit group $(x_{16s+3}^{r+1}, x_{16s+7}^{r+1}, x_{16s+11}^{r+1}, x_{16s+15}^{r+1})$

where $0 \leq s \leq 7$. We denote $\mathcal{C}_s$ the set that contains the candidates of the 16 key bits whose positions are in $\mathcal{K}'_s$ for $0 \leq s \leq 7$. At the beginning, $\mathcal{C}_s$ has in total 2^{16} candidates.

A practical 9-round attack (Utilizing a 7-round distinguisher). This attack employs the 7-round integral distinguisher ID_7 as illustrated in Table 8. Specifically, we have $r = 7$, $d = 14$, and $b_s = 4$ for all $0 \leq s \leq 7$. To ensure that only 2^4 candidates remain in each $\mathcal{C}_s$, we need to filter each $\mathcal{C}_s$ three times. Note that for a fixed s, each filtering of $\mathcal{C}_s$ requires 2^{14} different chosen plaintexts, but for different s, the same chosen plaintexts can be reused. Consequently, the data complexity of this attack is $3 \times 2^{14} \approx 2^{15.585}$.

Regarding the time complexity, we must first evaluate the cost of one filtering round, which involves two parts: partial decryptions and XOR sum computations. In this attack, we consider a 9-round BAKSHEESH as a unit of time. For each $s \in \{0, 1, \cdots, 7\}$, it takes $16 + 2 \times 4 = 24$ bitwise XORs and 4 lookup-table operations (for the 4-bit inverse SBox) to obtain the information for the s-th 4-bit group $(x^{r+1}_{16s+3}, x^{r+1}_{16s+7}, x^{r+1}_{16s+11}, x^{r+1}_{16s+15})$ using a guessed key and known X^{10}. We have tested the latency of a bitwise XOR and a lookup-table on a typical PC. The experimental results indicate that the latency of a bitwise XOR is nearly equal to that of a lookup-table. For simplicity, we regard a lookup-table as a bitwise XOR. Thus, the cost of the above partial decryption can be converted to 28 bitwise XORs. Additionally, an XOR sum computation requires $2^{14} - 1$ bitwise XORs. Therefore, the total cost of a 9-round...

BAKSHEESH contains 128×10 bitwise XORs and 32×9 lookup-table operations, which can be integrated into 1568 bitwise XORs. Now, let us compute the 9-round time complexity that is composed of the following parts:

(a) To filter guessed values by integral distinguisher, we need to prepare $2^{14} \times 3 \times X^{10}$.

(b) To filter $\mathcal{C}_s$ for each $s \in \{0, \ldots, 7\}$ 3 times to get 2^4 candidates, we need $2^{14} \times (2^{16} + 2^{12} + 2^8) \approx 2^{30.093}$ partial decryptions and $2^{16} + 2^{12} + 2^8 \approx 2^{16.093}$ XOR sum computations at 4 bits, which can be transformed to $[2^{30.093} \times 28 + 2^{16.093} \times (2^{14} - 1) \times 4]/1568 \approx 2^{24.478}$.

(c) To determine the remaining 32-bit information, we need an additional 2^{32} 9-round encryptions.

Summing up all the parts, the time complexity to recover 128 key bits is

$$2^{14} \times 3 + 2^{24.478} \times 8 + 2^{32} \approx 2^{32.009}.$$

Apparently, the time complexity is mainly caused by filtering the remaining 2^{32} candidates. Both the time and data complexity of this 9-round attack are practical. Actually, on the platform with an i7-8700 CPU @ 3.20GHz and 24 GB RAM, it takes at most 62 minutes to recover all the key bits using a single thread.

In addition to the 9-round practical attack described above, we also have:

- A 10-round key recovery attack.
- A 13-round theoretical key recovery attack.

5 Automated Cryptanalysis

5.1 Constraints for S-Box to be Active

To ensure the S-Box is active while any of the input bits is active:

$$\begin{aligned} x_{00} - a_{00} &\leq 0, \\ x_{01} - a_{00} &\leq 0, \\ x_{02} - a_{00} &\leq 0, \\ x_{03} - a_{00} &\leq 0. \end{aligned}$$

If the S-Box a_{ij} is active, any one of the input bits x_{ij} must be active:

$$x_{00} + x_{01} + x_{02} + x_{03} - a_{00} \geq 0.$$

Finally, an input difference must result in an output difference:

$$\begin{aligned} 4x_{10} + 4x_{11} + 4x_{12} + 4x_{13} - x_{00} - x_{01} - x_{02} - x_{03} &\geq 0, \\ 4x_{00} + 4x_{01} + 4x_{02} + 4x_{03} - x_{10} - x_{11} - x_{12} - x_{13} &\geq 0. \end{aligned}$$

5.2 How to Integrate S-Box into Linear Constraints?

We need the equations in **Algebraic Normal Form (ANF)** for the S-Box:

$$\begin{aligned} y_0 &= x_0x_2 \oplus x_0 \oplus x_1 \oplus x_3 \oplus 1, \\ y_1 &= x_0 \oplus x_1x_2 \oplus x_3 \oplus 1, \\ y_2 &= x_0x_2 \oplus x_1x_2 \oplus x_1 \oplus x_3, \\ y_3 &= x_0x_1 \oplus x_0x_2 \oplus x_2 \oplus x_3. \end{aligned}$$

Example: Constraints for y_1

To compute $y_1 = x_0 \oplus x_1x_2 \oplus x_3 \oplus 1$, we need constraints for **AND** and **XOR** operations.

1. Constraints for AND Operation For $z = x_{01}x_{02}$:

$$\begin{aligned} x_{01} + x_{02} - 1 &\leq z, \\ x_{01} &\geq z, \\ x_{02} &\geq z. \end{aligned}$$

2. Constraints for XOR Operation For $z_1 = z \oplus x_{00}$:

$$\begin{aligned} z - x_{00} &\leq z_1, \\ x_{00} - z &\leq z_1, \\ x_{00} + z &\geq z_1, \\ 2 - x_{00} - z &\geq z_1. \end{aligned}$$

For $z_2 = z_1 \oplus x_{03}$:

$$\begin{aligned}
&z_1 - x_{03} \leq z_2,\\
&x_{03} - z_1 \leq z_2,\\
&x_{03} + z_1 \geq z_2,\\
&2 - x_{03} - z_1 \geq z_2.
\end{aligned}$$

Finally, for $z_3 = z_2 \oplus 1$:

$$\begin{aligned}
&2 - 1 - z_2 \geq z_3,\\
&z_2 - 1 \leq z_3,\\
&1 + z_2 \geq z_3,\\
&2 - 1 - z_2 \geq z_3.
\end{aligned}$$

Thus, $y_1 = z_3 = x_{01}x_{02} \oplus x_{00} \oplus x_{03} \oplus 1$.

Big Endian and Little Endian

Endianness refers to the order in which bytes of a multi-byte data type (like integers or floating-point numbers) are stored in the memory. The two types of Endians are:

- **Big Endian:** The most significant byte (MSB) is stored at the lowest memory address.
- **Little Endian:** The least significant byte (LSB) is stored at the lowest memory address.

Explaining with the help of an example :

Consider a 32-bit integer value:

`0x12345678`

Big Endian Representation

In a big-endian system, the bytes are stored in memory as follows:

Memory Address	Value
0x00	12
0x01	34
0x02	56
0x03	78

Little Endian Representation

In a little-endian system, the bytes are stored in memory as follows:

Memory Address	Value
0x00	78
0x01	56
0x02	34
0x03	12

C Code for the Demonstration Purpose :

Here is a simple C program to demonstrate endianness:

Listing 1: C Code to Demonstrate Endianness

```c
#include <stdio.h>

int main() {
    unsigned int num = 0x12345678; // Example number
    unsigned char *ptr = (unsigned char*)&num;

    printf("Memory representation:\n");
    for (int i = 0; i < sizeof(num); i++) {
        printf("Address %p: 0x%x\n", ptr + i, *(ptr + i));
    }

    return 0;
}
```

Output

- On a **Little Endian** system:

```
Address 0x100: 0x78
Address 0x101: 0x56
Address 0x102: 0x34
Address 0x103: 0x12
```

- On a **Big Endian** system:

```
Address 0x100: 0x12
Address 0x101: 0x34
Address 0x102: 0x56
Address 0x103: 0x78
```

6 Github

Link

References

1. https://eprint.iacr.org/2023/750.pdf

2. https://eprint.iacr.org/2024/1926.pdf