Exploring the LED Cipher

Dhruv Deshmukh[1], Jyotika Mahapatra[1] and Nikhil Chaudhary[1]

Indian Institute of Technology, Bhilai, India
dhruvr@iitbhilai.ac.in
jyotikam@iitbhilai.ac.in
nikhilch@iitbhilai.ac.in

Abstract. In this paper we analyse the LED Cipher.
We discuss the construction of LED Cipher and try to illustrate it using various diagrams. We further analyse its Sbox and crypt analyze the LED Cipher. Finally we briefly discuss about the implementation and application developed using the cipher.

Keywords: Lightweight · block cipher · cryptanalysis

1 Introduction

With the onset of IoT there has been increase in number of small devices with computational powers. These small devices are a part of larger systems and communicate with each other. There is a need for cipher implementations that can be used in these devices for security of communication and storage. Since the devices have limited computational power and memory the cipher implementations need to be lightweight. Light Encryption Device, *LED* is one such cipher that is based on the AES architecture but encodes smaller block size of 64-bits instead of the standard size of 128-bits. In our term paper we investigate this cipher and produce our results.

The term paper consists the analysis of the DDT and LAT for the S-box used in the cipher and their properties like differential-uniformity, differential branch number. We then present the differential and integral cryptanalysis as well as the try to model automated cryptanalysis for one round. Throughput the paper we have endeavoured to represent everything in form of neat diagrams that make it easy to understand our work. Apart from this you can find the implementation of this cipher done by us in Python on github. We also have provided different modes of operation like cbc, ctr, etc in our implementation. Finally, we have built a voice chat application in Python that uses the LED cipher for encryption and decryption.

The major differences between AES and LED are in the block size encoded, the key schedule, the mix columns operation and the use of round constants. LED has implemented these components such that the cipher is lightweight and uses hardware more efficiently. We will go over these differences in detail in the next session where we explain the design of this cipher.

2 Design and Specification

One round of cipher encryption includes AddRoundKey and then the round function. The round function consists of four steps which are:

1. Add Constants
2. Sub Cells

3. Shift Rows

4. Mix Columns serial

Below is the overview of the 4 steps:

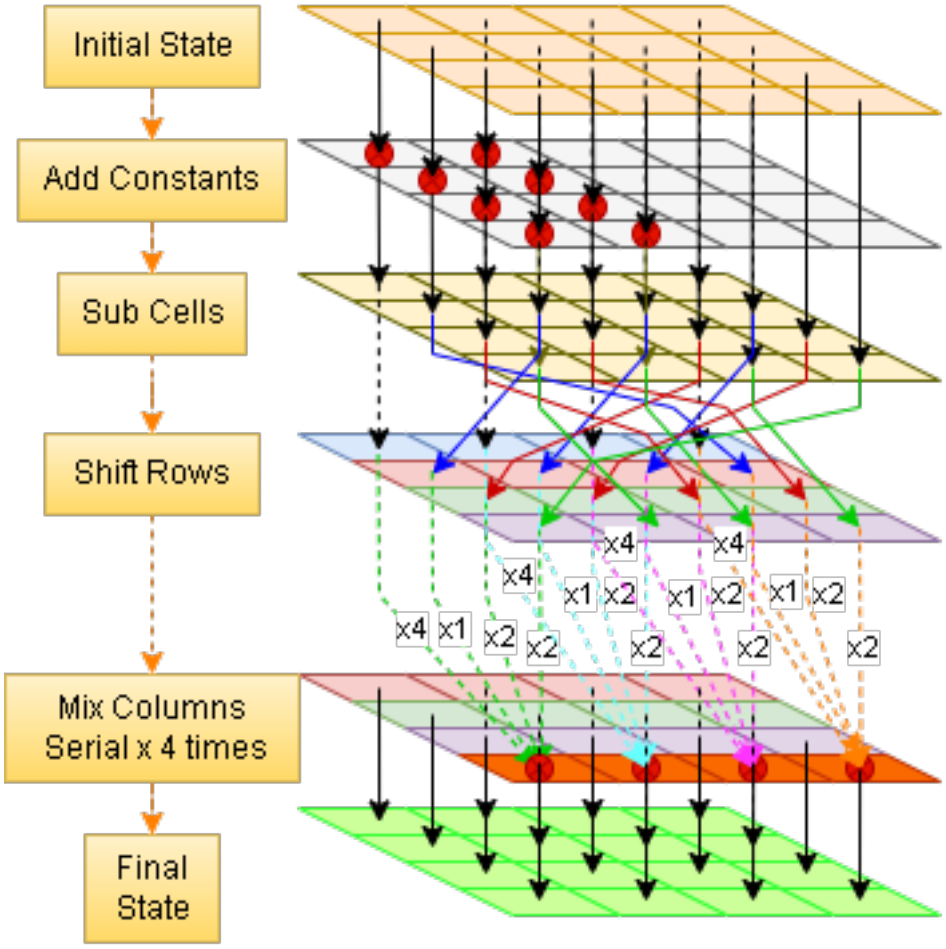
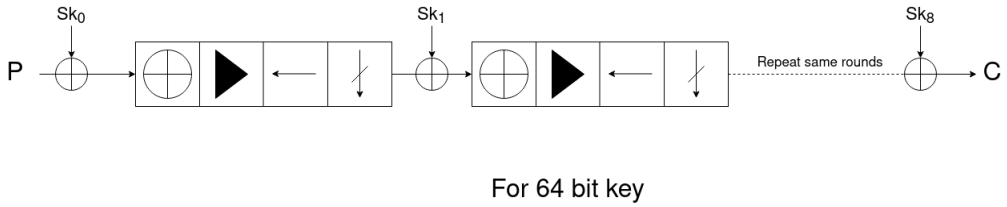
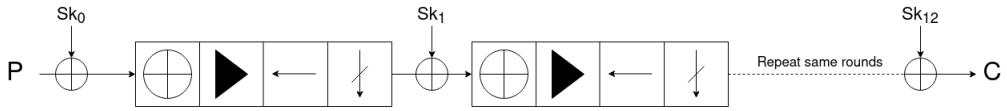


Figure 1: 4 steps in the round function of LED

The number of rounds depends on the key-size. For 64 bit keys the total number of rounds is 8 and for 128 bits number of rounds is 12.



For 64 bit key



For 128 bit key

Before going into the details of these steps we need to get know how the sub-keys for each round are obtained so let us understand the key schedule. The key schedule is quite simplistic in LED. Here we give the formula for generating j-th cell of i-th sub-key:

sk_0^i	sk_1^i	sk_2^i	sk_3^i
sk_4^i	sk_5^i	sk_6^i	sk_7^i
sk_8^i	sk_9^i	sk_{10}^i	sk_{11}^i
sk_{12}^i	sk_{13}^i	sk_{14}^i	sk_{15}^i

$$sk_j^i = k_{(j+i*16 \ mod \ l)}$$

where **k** is the initial key and **l** is the length of the key in nibbles. Using this we show the sub-keys obtained for a 64-bit and 128-bit initial keys.

k_0	k_1	k_2	k_3
k_4	k_5	k_6	k_7
k_8	k_9	k_{10}	k_{11}
k_{12}	k_{13}	k_{14}	k_{15}

sub-keys for 64-bit initial key

k_0	k_1	k_2	k_3	k_16	k_17	k_18	k_19
k_4	k_5	k_6	k_7	k_20	k_21	k_22	k_23
k_8	k_9	k_{10}	k_{11}	k_24	k_25	k_{26}	k_{27}
k_{12}	k_{13}	k_{14}	k_{15}	k_{28}	k_{29}	k_{30}	k_{31}

sub-keys for 128-bit initial key

Thus in the 64-bit case all the sub-keys will be same and equal to the initial key provided while in case of 128-bit key the sub-keys will alternate between the 16 MSB of key and 16 LSB of the key depending on the round number **i**. The thing to note here is that the key bits are organized in the row major order instead of the column major order used in the AES and the same is true for the message as well. This is done considering that the row major form is more hardware friendly. Thus the plaintext matrix can be visualized as:

m_0	m_1	m_2	m_3
m_4	m_5	m_6	m_7
m_8	m_9	m_{10}	m_{11}
m_{12}	m_{13}	m_{14}	m_{15}

64-bit plaintext **m**, $m_0||m_1||...||m_{14}||m_{15}$

The **AddRoundKey** operation just XORs i-th round sub-key with the state matrix and the result is given below:

$sk_0^i \bigoplus m_0$	$sk_1^i \bigoplus m_1$	$sk_2^i \bigoplus m_2$	$sk_3^i \bigoplus m_3$
$sk_4^i \bigoplus m_4$	$sk_5^i \bigoplus m_5$	$sk_6^i \bigoplus m_6$	$sk_7^i \bigoplus m_7$
$sk_8^i \bigoplus m_8$	$sk_9^i \bigoplus m_9$	$sk_{10}^i \bigoplus m_{10}$	$sk_{11}^i \bigoplus m_{11}$
$sk_{12}^i \bigoplus m_{12}$	$sk_{13}^i \bigoplus m_{13}$	$sk_{14}^i \bigoplus m_{14}$	$sk_{15}^i \bigoplus m_{15}$

Add constants: The round constant is used differently in the LED construction. Instead of using it in the key schedule the round constant is directly XORed with the state in this step. The matrix XORed with the state in this step is given below:

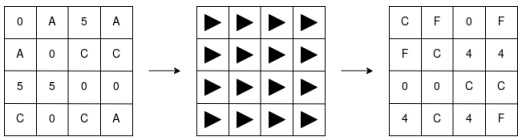
$0 \bigoplus (ks_7 \| \| ks_6 \| \| ks_5 \| \| ks_4)$	$rc_5^i \| \| rc_4^i \| \| rc_3^i$	0	0
$1 \bigoplus (ks_7 \| \| ks_6 \| \| ks_5 \| \| ks_4)$	$rc_2^i \| \| rc_1^i \| \| rc_0^i$	0	0
$2 \bigoplus (ks_3 \| \| ks_2 \| \| ks_1 \| \| ks_0)$	$rc_5^i \| \| rc_4^i \| \| rc_3^i$	0	0
$3 \bigoplus (ks_3 \| \| ks_2 \| \| ks_1 \| \| ks_0)$	$rc_2^i \| \| rc_1^i \| \| rc_0^i$	0	0

The ks is the key size in bits. We divide it into 4 MSB and 4 LSB and use them in first column in matrix shown above. The initial value of round constant is 0 then for subsequent rounds we generate the constant as follows. Shift current round constant left by one and new value of bit $rc_0 = rc_5 \bigoplus rc_4 \bigoplus 1$

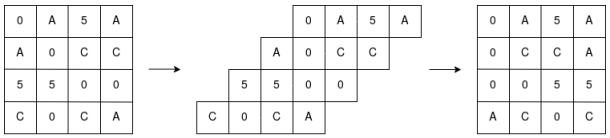
$$(rc_5^i \| \| rc_4^i \| \| rc_3^i \| \| rc_2^i \| \| rc_1^i \| \| rc_0^i) = (rc_4^{i-1} \| \| rc_3^{i-1} \| \| rc_2^{i-1} \| \| rc_1^{i-1} \| \| rc_0^{i-1} \| \| rc_5^{i-1} \oplus rc_4^{i-1}) \oplus 1$$

Sub Cells: For this operation the nibbles in each cell are substituted using the S-box given below.

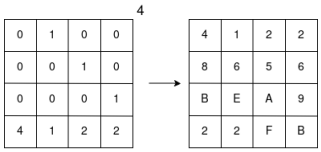
x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2



Shift Rows: The shift rows operation is exactly same as the AES shift rows with i-th row shifting left by i places in a circular fashion. The diagram is given below for better visualization:



Mix Columns Serial: The concept of this operation is same as AES mix columns but the matrix is used is different. Also, keeping in mind the hardware efficiency the implementation is not just matrix multiplication but instead the matrix chosen is such that it is the 4 power of a special matrix which is easy to implement in hardware. This special matrix is thus applied 4 times in succession to get the desired output. The matrix is given below. One application of this matrix shifts all columns up by one and puts a linear combination of all elements in a column as the new last element of that column. When this is done 4 times we get a state that is mixed well.



3 Analysis of the DDT and LAT and comparison with other ciphers

DDT

in/out	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	0	4	0	4	0	0	0	4	0	0
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0
3	0	2	0	2	2	0	4	2	0	0	2	2	0	0	0	0
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0
5	0	2	0	0	2	0	0	0	0	2	2	2	4	2	0	0
6	0	0	2	0	0	0	2	0	2	0	0	4	2	0	0	4
7	0	4	2	0	0	0	2	0	2	0	0	0	2	0	0	4
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4
9	0	0	2	0	4	0	2	0	2	0	0	0	2	0	4	0
A	0	0	2	2	0	4	0	0	2	0	2	0	0	2	2	0
B	0	2	0	0	2	0	0	0	4	2	2	2	0	2	0	0
C	0	0	2	0	0	4	0	2	2	2	2	0	0	0	2	0
D	0	2	4	2	2	0	0	2	0	0	2	2	0	0	0	0
E	0	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
F	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	4

LAT

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

Comparison with other ciphers

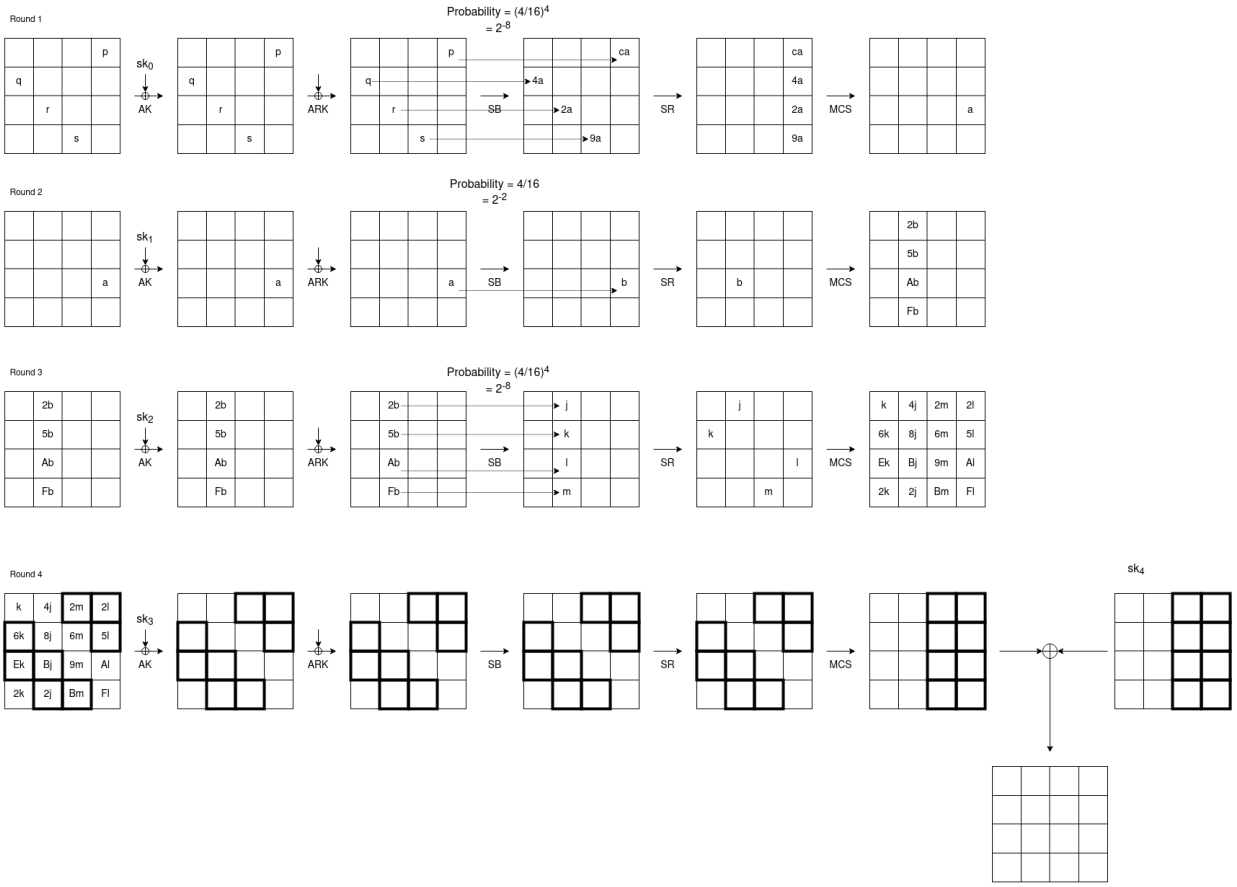
Cipher Name	SBox Size	Differential Uniformity	Differential Branch Number
Midori	4-bit	4	2
GIFT	4-bit	6	2
Serpent	4-bit	4	3
Prince	4-bit	4	2
Pride	4-bit	4	2
Ascon	5-bit	8	3
Klein	4-bit	4	2
PHOTON-beetle	4-bit	4	3
LED	**4-bit**	**4**	**3**
Elephant	4-bit	4	3
Wage	8-bit	8	2
Aria	8-bit	4	2
Primates APE	5-bit	2	2
Skinny	4-bit	2	2
Print	3-bit	2	2

The values of our sbox seem to be in the middle among the 4-bit sboxes. Differential uniformity is 2,4 or 6 while branch number is 2 or 3.

4 Differential and Integral Cryptanalysis

4.1 Differential Cryptanalysis

For differential cryptanalysis, 2 plaintexts are taken such that the difference activates the cells as follows.



First 2 columns of sk_5 is guessed and the states are retraced back as shown above. The selected cells are then used to check if they follow the same pattern. If not then the guess can be discarded.

4.2 Integral Cryptanalysis

The set $P = \{P_0, P_1, ...P_{15}\}$ such that the first nibble of each plain text is unique and the set covers all possible values. Therefore the set P follows property as follows :

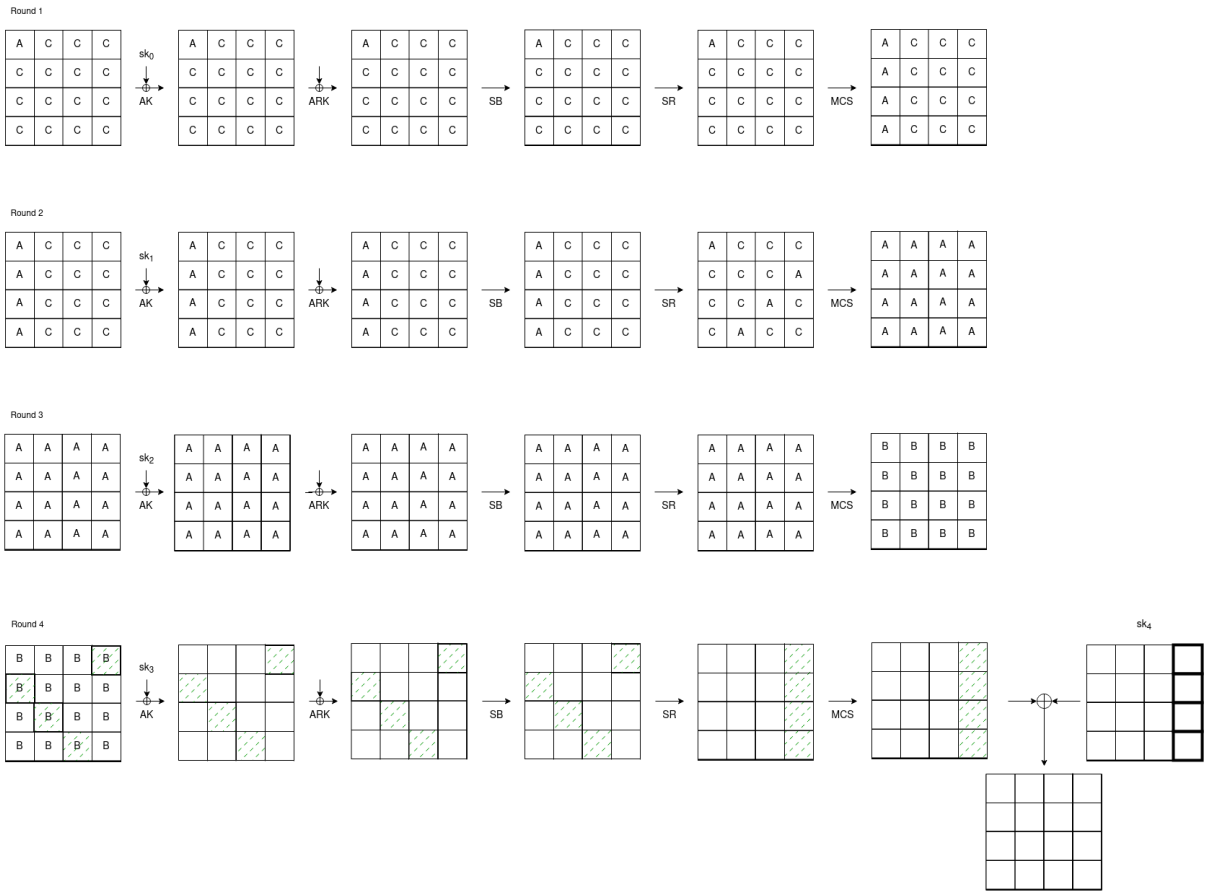
A	C	C	C
C	C	C	C
C	C	C	C
C	C	C	C

Where A is the all property and C is constant.
Similar to AES the A and C property is preserved after XOR with round key. As Round constant addition too is XORing bitwise, the properties remain the same. The results are same for Sub-cells and Shift rows too as they were in the case of AES. For Mix column serial we observe the following for P_i where i = 0 to 15 :

$$\begin{bmatrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{bmatrix} \begin{bmatrix} i \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 4i \wedge C_1 \wedge 2C_2 \wedge 2C_3 \\ 8i \wedge 6C_1 \wedge 5C_2 \wedge 6C_3 \\ Bi \wedge EC_1 \wedge AC_2 \wedge 9C_3 \\ 2i \wedge 2C_1 \wedge FC_2 \wedge BC_3 \end{bmatrix}$$

$$= \begin{bmatrix} 4i \\ 8i \\ Bi \\ 2i \end{bmatrix} \wedge \begin{bmatrix} C_1 \wedge 2C_2 \wedge 2C_3 \\ 6C_1 \wedge 5C_2 \wedge 6C_3 \\ EC_1 \wedge AC_2 \wedge 9C_3 \\ 2C_1 \wedge FC_2 \wedge BC_3 \end{bmatrix}$$

As i covers the All property and 4i, 8i, Bi, 2i vary to take all possible 16 values, the entire column gains All property. In the third round after the MCS, it is observed that Balanced property is acquired similar to AES since the only difference during XOR are the constants multiplied which do not have a influence over all.



First 4 nibbles of sk_4. The Balanced property should hold true at the respective positions as shown in figure-2, this can be used as the distinguish-er too.
Probability of the 4 positions being balanced is $(2^{-4})^4$. With 2^{16} guesses, number of guesses passing the condition is 1, thus the sub-key space reduces to 1.

Data Complexity :
1 set of 2^4 plain text required. Thus the data complexity is 2^4 chosen plain texts.
Time Complexity:
1 round of decryption is done for each 2^16 guesses and 2^4 cipher texts. Thus the computational cost $= 1 * 2^{16} * 2^4 = 2^{20}$ round functions. This is equivalent to $\frac{2^{20}}{4} = 2^{18}$ four round LED computation.
It is repeated for the remaining three columns. Thus total time complexity is $4 * 2^{18} = 2^{20}$ 4 round LED computations.
Memory:
As the subkey space gets reduced to 1 in a single iteration thus the memory complexity is negligible.

5 Automated Cryptanalysis

The main thing we have to model constraints for is the SBOX as the add constants and add round key do not change the output difference. In our case the state is of 16 nibbles and in the SubCells operation they get substituted using the SBOX. Hence in our case we constraints for 16 sboxes each of which has 4 input bits and 4 output bits. Let the 16 sboxes be given variables $A_{i,j}$ where **i** is for denoting the round number and **j** is the position of sbox within the state. The variable is one when it receives non-zero input difference i.e. the sbox is active and 0 otherwise. Let input bits of the state be denoted by $x^{in}_{i,k}$ and $x^{out}_{i,k}$ where i denotes the round number, k denotes the bit number ranging from 0 to 63 and input/output stand for input/output respectively. Now for an sbox $A_{i,j}$ the inputs and outputs are $x^{in}_{i,4j+m}$ and $x^{out}_{i,4j+m}$ where m ranges from 0 to 3. Now we will show the sbox constraints for i-th round and j-th sbox and same will be applicable for all the sboxes. The constraints are as follows:

1. Sbox is active if atleast one of the input bits to it are active

$$x^{in}_{i,4j+m} - A_{i,j} \leq 0 \, \forall m \in \{1, 2, 3, 4\}$$

2. When sbox is active then one of the input bits is one

$$x^{in}_{i,4j+0} + x^{in}_{i,4j+1} + x^{in}_{i,4j+2} + x^{in}_{i,4j+3} - A_{i,j} \geq 0$$

3. Input difference must result in output difference and vice versa

$$x^{out}_{i,4j+0} + 4x^{out}_{i,4j+1} + 4x^{out}_{i,4j+2} + 4x^{out}_{i,4j+3} - x^{in}_{i,4j+0} + x^{in}_{i,4j+1} + x^{in}_{i,4j+2} + x^{in}_{i,4j+3} \geq 0$$

$$x^{in}_{i,4j+0} + 4x^{in}_{i,4j+1} + 4x^{in}_{i,4j+2} + 4x^{in}_{i,4j+3} - x^{out}_{i,4j+0} + x^{out}_{i,4j+1} + x^{out}_{i,4j+2} + x^{out}_{i,4j+3} \geq 0$$

Also the input bit now will go through the shift rows and mix columns operation and then go to second round. Thus each input bit for a round is the linear combination of output bits from the previous round. For example let us write this relation for the first input bit for i-th round:

$$x^{in}_{i,0} = x^{out}_{i-1,2} \oplus x^{out}_{i-1,20} \oplus x^{out}_{i-1,41} \oplus x^{out}_{i-1,61}$$

Similarly we can write for other bits but the relations will be more complicated. Thus keeping the above constraints in mind we can model automated cryptanalysis for one round and then repeat them for the subsequent rounds.

6 Brief on Software implementation and application

We have implemented the cipher in Python. Apart from just the block encryption and decryption functions we have provided functions for different modes of operation like cbc, pcbc, cfb, ofb, and ctr. Also, we have function for iv generation. Finally we use hmac to ensure integrity of message. Using this cipher we have made simple vault application. In this there is folder called vault. The files in this folder are encrypted and only the person with the key will able to decrypt them and view them.

7 Conclusion

In this paper we explored about the LED cipher. We made an effort to make its design clear by explaining the steps through the use of multiple diagrams. We analysed the DDT and LAT of the our cipher and used it for trying out differential and integral cryptanalysis on it. We also compared the DDT with that of other ciphers on the basis of properties like differential branch number and uniformity and found our cipher had decent values. IN automated cryptanalysis section we tried to model one round of the cipher using an MILP optimisation problem. Finally we briefly dicussed about our software and application implementation.

8 References

1. LED paper
2. Our software implementation