Term Paper - LED

Aniket Raj[1], Kumar Shivendu[1] and Aastha Asthana[1]

Indian Institute of Technology Bhilai, {aniketr,shivendu,aastha}@iitbhilai.ac.in

Abstract. In this term paper, LED block cipher is discussed in detail. In particular, we have presented the design specifications of the cipher along with a brief overview of its software and hardware implementations. The comparison of the performance between the software and hardware implementations is also presented to demonstrate the feasibility and how it stands out from AES. In the end, we discuss the security and cryptanlysis of the cipher with different attack methods. A special emphasis has been laid out to the fault attack model of LED along with its implementation.

Keywords: LED, lightweight, block cipher, AES · Fault Attack

1 Introduction

AES and the ciphers based on it are often optimised for fast software implementations which generally lags behind in providing a light implementation in hardware. In the past few years, we have seen a huge surge in Internet of Things(IoT) and highly-constrained devices. This paves the way for a lightweight block cipher which is compact enough for hardware and still efficient in software implementation. This is where LED comes into the picture. In addition to the above primary requirement, it serves another two key goals : (a) A very light key scheduling (b) resistance to related and single-key attacks.

In general we can say that most of the lightweight block ciphers are vulnerable to key-related attacks. For instance, HIGHT has a known related-key attack (K+-2010). Hummingbird-1 and KTANTAN are known to have practical related-key attacks(S-2011 and A-2011 respectively). However, LED is resistant from any such attacks in spite of having a practically non existent key-schedule.

2 Design Specifications

The variant discussed here is a 64-bit LED cipher which can take instances of 64-bit or 128-bit key size. However, we will limit our discussion to 64-bit keys only for the sake of simplification. Just like AES, here we'll have a $4 * 4$ state of LED.

LED uses the the similar basic operations in a round as that of the AES. The basic operations along with their description is given below :

1. AddConstants: For any single round, this is the first step where we xor a round-dependent constant in the first two columns of the state. The size bits of round constants are initialized to all zeroes and they are shifted one position to the left. For instance, rc_0 is computed as rc_5 *xor* rc_4 *xor* 1. We label the 8 bits representing the key size in bits - $ks_7, ..., ks_0$. For a given round, the round constants are arranged in the following way

$$\begin{bmatrix} 0 \oplus (\mathrm{ks}_7 \| \mathrm{ks}_6 \| \mathrm{ks}_5 \| \mathrm{ks}_4) & (\mathrm{rc}_5 \| \mathrm{rc}_4 \| \mathrm{rc}_3) & 0 & 0 \\ 1 \oplus (\mathrm{ks}_7 \| \mathrm{ks}_6 \| \mathrm{ks}_5 \| \mathrm{ks}_4) & (\mathrm{rc}_2 \| \mathrm{rc}_1 \| \mathrm{rc}_0) & 0 & 0 \\ 2 \oplus (\mathrm{ks}_3 \| \mathrm{ks}_2 \| \mathrm{ks}_1 \| \mathrm{ks}_0) & (\mathrm{rc}_5 \| \mathrm{rc}_4 \| \mathrm{rc}_3) & 0 & 0 \\ 3 \oplus (\mathrm{ks}_3 \| \mathrm{ks}_2 \| \mathrm{ks}_1 \| \mathrm{ks}_0) & (\mathrm{rc}_2 \| \mathrm{rc}_1 \| \mathrm{rc}_0) & 0 & 0 \end{bmatrix}$$

2. `SubCells:` For the purpose of confusion, every nibble in the state is replaced using the sbox of `LED` which is identical to that of `PRESENT`.

3. `Shiftrows:` Here permutation is done by shifting rows. The nibbles are rotated such that the i_{th} line is pushed by i positions onto the left.

4. `MixColumns Serial:` For the purpose of diffusion, the mixcolumn is performed but this operation is not identical to that of the `AES`. Here we apply the special construction of the MDS matrix to every column independently. This operation can be seen as applying a hardware-friendly matrix A (no additional memory cell is needed both for encryption and decryption from hardware's perspective)four times in order to get the MDS matrix M.

$$(A)^4 = \begin{pmatrix} 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \\ 4\,1\,2\,2 \end{pmatrix}^4 = \begin{pmatrix} 4\,1\,2\,2 \\ 8\,6\,5\,6 \\ \mathrm{B}\,\mathrm{E}\,\mathrm{A}\,9 \\ 2\,2\,\mathrm{F}\,\mathrm{B} \end{pmatrix} = M.$$

Pipeline of the Encryption

In this section, we will walk through the entire pipline of the encryption starting from the plaintext, P which would involve the above basic operations. In the beginning, we will have the state initialized with the message, m. For instance, if we have a 64-bit plaintext, the 16 nibbles will be arranged as shown below in a row wise manner instead of column-wise manner which has been proven to be more hardware friendly as argued in [3].

$$\begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix}$$

A round in LED means applying the four basic operations as discussed above - AddConstants, SubCells, ShiftRows and MixColumns. The repetitions of these rounds 4 times is called a `step`. It is interesting to note here that this basic unit `step` consisting of 4 rounds does not involve the key. We have another operation called `addRoundKey` which xors the key with the state before going into any `step`. This can be visualized in the diagram shown below.

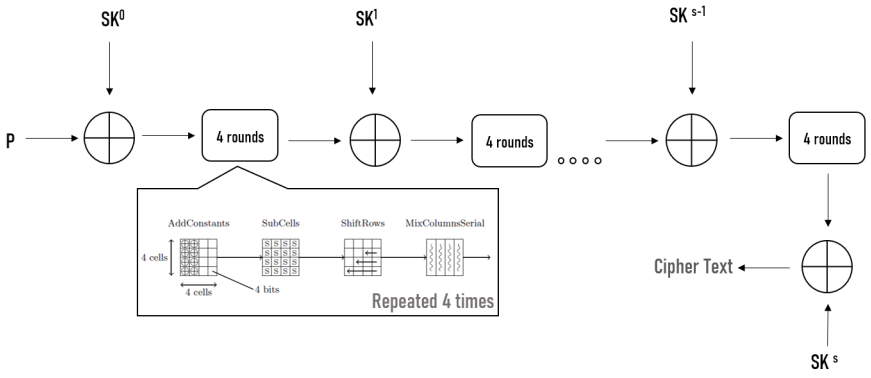


Figure 1: LED Design

The operation $addRoundKey(state, SK_i)$ xors the elements of the subkey SK_i with the given state in the respective array positions. The i_{th} subkey SK^i is derived from the key (consisting of l nibbles) using $sk^i_j = k_{j+i*16 \ mod \ l}$. The convention followed in LED is that the number of steps s during encryption should be 8 for 64-bit key and for bigger key sizes up to 128 bits, it must be 12.

3 Security Analysis

As mentioned in the abstract, one of the primary motivation of LED was the resistance against related key attacks. This is the major reason why total number of rounds is largely conservative when compared to AES. For instance, the 64-bit variant has a total of 32 AES-like rounds (8 steps * 4 rounds). To briefly demonstrate this, we will take help of the well known 4 rounds proof for AES which shows that there are at least 25 active sboxes in it. Since one step consists of four rounds, we can say that any differential path of LED will have at least 25 * s active s-boxes.

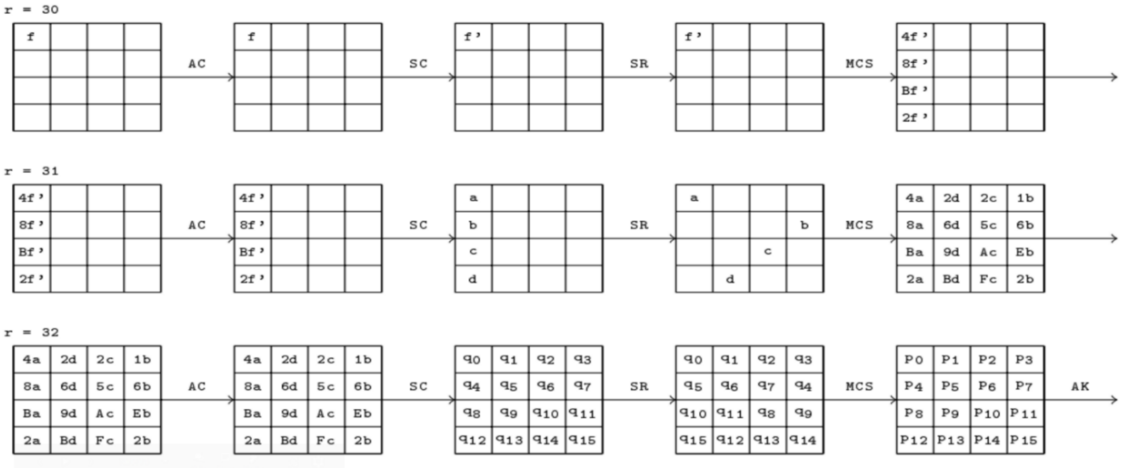
The simple or non-existent key scheduling algorithm as stated above becomes before useful to set upper bounds on the number of active s boxes even under the setting of related key. Let us consider the related key model with 64-bit key size. Here the key need not be divided into 2 or more independent parts. Therefore, every subkey will be active and for every 2 steps, one will definitely be active. Thus any related key differential path will have at least $s/2 * 25$ active sboxes. The same argument is also true for 128-bit key variant.

In terms of other known crptanalysis techniques, LED is practically resistant to them due to its conservative approach in the number of rounds. In integral attacks with known-key model, it reaches only 7 rounds with a time complexity of 2^{28}. The same goes for zero-sum partitions attack where there is a distinguisher for at most 12 rounds with a time complexity of 2^{64}. Also, the classical differential attack shows that we have the upper bound of 2^{-32} on the best differential path probability on 4 active rounds of LED using the results shown by [4]. In this paper we will focus only on a very interesting fault attack on LED which is explained in the following section along with its implementation.

4 Fault Attack

We have implemented a Fault attack on LED64 which consists of 32 rounds. We assume that the attacker is capable of injecting a fault of 4 bit at the first element of the state matrix in the 30^{th} round. We observe how the fault is spreading in the last 3 rounds and

find a way to easily recover the key. The fault spreads as:



The XOR difference derived from the expressions of C and C'(faulty) is computed and compared with the corresponding fault value. To derive the fault equations, we need to invert each round.The inverse calculation is as follows:

(**AddKey**)$^{-1}$: $c_i + k_i$ and $c'_i + k_i$
(**MixColumnsSerial**)$^{-1}$: The inverse matrix is calculated and then is multiplied with the state matrix. Inverse matrix is:

$$M^{-1} = \begin{pmatrix} \mathtt{C} & \mathtt{C} & \mathtt{D} & 4 \\ 3 & 8 & 4 & 5 \\ 7 & 6 & 2 & \mathtt{E} \\ \mathtt{D} & 9 & 9 & \mathtt{D} \end{pmatrix}$$

(**ShiftRows**)$^{-1}$:The nibbles are rotated such that the i^{th} line is pushed by i positions onto the right.

(**SubCells**)$^{-1}$: Inverse of the LED Sbox is taken.

After applying these,

Fault equations for a.

$$\begin{aligned}
4 \cdot a &= S^{-1}(\mathtt{C} \cdot (c_0 + k_0) + \mathtt{C} \cdot (c_4 + k_4) + \mathtt{D} \cdot (c_8 + k_8) + 4 \cdot (c_{12} + k_{12})) + \\
&\quad S^{-1}(\mathtt{C} \cdot (c'_0 + k_0) + \mathtt{C} \cdot (c'_4 + k_4) + \mathtt{D} \cdot (c'_8 + k_8) + 4 \cdot (c'_{12} + k_{12})) \qquad (E_{a,0}) \\
8 \cdot a &= S^{-1}(3 \cdot (c_3 + k_3) + 8 \cdot (c_7 + k_7) + 4 \cdot (c_{11} + k_{11}) + 5 \cdot (c_{15} + k_{15})) + \\
&\quad S^{-1}(3 \cdot (c'_3 + k_3) + 8 \cdot (c'_7 + k_7) + 4 \cdot (c'_{11} + k_{11}) + 5 \cdot (c'_{15} + k_{15})) \qquad (E_{a,1}) \\
\mathtt{B} \cdot a &= S^{-1}(7 \cdot (c_2 + k_2) + 6 \cdot (c_6 + k_6) + 2 \cdot (c_{10} + k_{10}) + \mathtt{E} \cdot (c_{14} + k_{14})) + \\
&\quad S^{-1}(7 \cdot (c'_2 + k_2) + 6 \cdot (c'_6 + k_6) + 2 \cdot (c'_{10} + k_{10}) + \mathtt{E} \cdot (c'_{14} + k_{14})) \qquad (E_{a,2}) \\
2 \cdot a &= S^{-1}(\mathtt{D} \cdot (c_1 + k_1) + 9 \cdot (c_5 + k_5) + 9 \cdot (c_9 + k_9) + \mathtt{D} \cdot (c_{13} + k_{13})) + \\
&\quad S^{-1}(\mathtt{D} \cdot (c'_1 + k_1) + 9 \cdot (c'_5 + k_5) + 9 \cdot (c'_9 + k_9) + \mathtt{D} \cdot (c'_{13} + k_{13})) \qquad (E_{a,3})
\end{aligned}$$

Fault equations for d.

$$\begin{aligned}
\mathtt{2}\cdot d = {} & S^{-1}(\mathtt{C}\cdot(c_1+k_1)+\mathtt{C}\cdot(c_5+k_5)+\mathtt{D}\cdot(c_9+k_9)+\mathtt{4}\cdot(c_{13}+k_{13}))+ \\
& S^{-1}(\mathtt{C}\cdot(c'_1+k_1)+\mathtt{C}\cdot(c'_5+k_5)+\mathtt{D}\cdot(c'_9+k_9)+\mathtt{4}\cdot(c'_{13}+k_{13})) \qquad (E_{d,0}) \\
\mathtt{6}\cdot d = {} & S^{-1}(\mathtt{3}\cdot(c_0+k_0)+\mathtt{8}\cdot(c_4+k_4)+\mathtt{4}\cdot(c_8+k_8)+\mathtt{5}\cdot(c_{12}+k_{12}))+ \\
& S^{-1}(\mathtt{3}\cdot(c'_0+k_0)+\mathtt{8}\cdot(c'_4+k_4)+\mathtt{4}\cdot(c'_8+k_8)+\mathtt{5}\cdot(c'_{12}+k_{12})) \qquad (E_{d,1}) \\
\mathtt{9}\cdot d = {} & S^{-1}(\mathtt{7}\cdot(c_3+k_3)+\mathtt{6}\cdot(c_7+k_7)+\mathtt{2}\cdot(c_{11}+k_{11})+\mathtt{E}\cdot(c_{15}+k_{15}))+ \\
& S^{-1}(\mathtt{7}\cdot(c'_3+k_3)+\mathtt{6}\cdot(c'_7+k_7)+\mathtt{2}\cdot(c'_{11}+k_{11})+\mathtt{E}\cdot(c'_{15}+k_{15})) \qquad (E_{d,2}) \\
\mathtt{B}\cdot d = {} & S^{-1}(\mathtt{D}\cdot(c_2+k_2)+\mathtt{9}\cdot(c_6+k_6)+\mathtt{9}\cdot(c_{10}+k_{10})+\mathtt{D}\cdot(c_{14}+k_{14}))+ \\
& S^{-1}(\mathtt{D}\cdot(c'_2+k_2)+\mathtt{9}\cdot(c'_6+k_6)+\mathtt{9}\cdot(c'_{10}+k_{10})+\mathtt{D}\cdot(c'_{14}+k_{14})) \qquad (E_{d,3})
\end{aligned}$$

Fault equations for c.

$$\begin{aligned}
\mathtt{2}\cdot c = {} & S^{-1}(\mathtt{C}\cdot(c_2+k_2)+\mathtt{C}\cdot(c_6+k_6)+\mathtt{D}\cdot(c_{10}+k_{10})+\mathtt{4}\cdot(c_{14}+k_{14}))+ \\
& S^{-1}(\mathtt{C}\cdot(c'_2+k_2)+\mathtt{C}\cdot(c'_6+k_6)+\mathtt{D}\cdot(c'_{10}+k_{10})+\mathtt{4}\cdot(c'_{14}+k_{14})) \qquad (E_{c,0}) \\
\mathtt{5}\cdot c = {} & S^{-1}(\mathtt{3}\cdot(c_1+k_1)+\mathtt{8}\cdot(c_5+k_5)+\mathtt{4}\cdot(c_9+k_9)+\mathtt{5}\cdot(c_{13}+k_{13}))+ \\
& S^{-1}(\mathtt{3}\cdot(c'_1+k_1)+\mathtt{8}\cdot(c'_5+k_5)+\mathtt{4}\cdot(c'_9+k_9)+\mathtt{5}\cdot(c'_{13}+k_{13})) \qquad (E_{c,1}) \\
\mathtt{A}\cdot c = {} & S^{-1}(\mathtt{7}\cdot(c_0+k_0)+\mathtt{6}\cdot(c_4+k_4)+\mathtt{2}\cdot(c_8+k_8)+\mathtt{E}\cdot(c_{12}+k_{12}))+ \\
& S^{-1}(\mathtt{7}\cdot(c'_0+k_0)+\mathtt{6}\cdot(c'_4+k_4)+\mathtt{2}\cdot(c'_8+k_8)+\mathtt{E}\cdot(c'_{12}+k_{12})) \qquad (E_{c,2}) \\
\mathtt{F}\cdot c = {} & S^{-1}(\mathtt{D}\cdot(c_3+k_3)+\mathtt{9}\cdot(c_7+k_7)+\mathtt{9}\cdot(c_{11}+k_{11})+\mathtt{D}\cdot(c_{15}+k_{15}))+ \\
& S^{-1}(\mathtt{D}\cdot(c'_3+k_3)+\mathtt{9}\cdot(c'_7+k_7)+\mathtt{9}\cdot(c'_{11}+k_{11})+\mathtt{D}\cdot(c'_{15}+k_{15})) \qquad (E_{c,3})
\end{aligned}$$

Fault equations for b.

$$\begin{aligned}
\mathtt{1}\cdot b = {} & S^{-1}(\mathtt{C}\cdot(c_3+k_3)+\mathtt{C}\cdot(c_7+k_7)+\mathtt{D}\cdot(c_{11}+k_{11})+\mathtt{4}\cdot(c_{15}+k_{15}))+ \\
& S^{-1}(\mathtt{C}\cdot(c'_3+k_3)+\mathtt{C}\cdot(c'_7+k_7)+\mathtt{D}\cdot(c'_{11}+k_{11})+\mathtt{4}\cdot(c'_{15}+k_{15})) \qquad (E_{b,0}) \\
\mathtt{6}\cdot b = {} & S^{-1}(\mathtt{3}\cdot(c_2+k_2)+\mathtt{8}\cdot(c_6+k_6)+\mathtt{4}\cdot(c_{10}+k_{10})+\mathtt{5}\cdot(c_{14}+k_{14}))+ \\
& S^{-1}(\mathtt{3}\cdot(c'_2+k_2)+\mathtt{8}\cdot(c'_6+k_6)+\mathtt{4}\cdot(c'_{10}+k_{10})+\mathtt{5}\cdot(c'_{14}+k_{14})) \qquad (E_{b,1}) \\
\mathtt{E}\cdot b = {} & S^{-1}(\mathtt{7}\cdot(c_1+k_1)+\mathtt{6}\cdot(c_5+k_5)+\mathtt{2}\cdot(c_9+k_9)+\mathtt{E}\cdot(c_{13}+k_{13}))+ \\
& S^{-1}(\mathtt{7}\cdot(c'_1+k_1)+\mathtt{6}\cdot(c'_5+k_5)+\mathtt{2}\cdot(c'_9+k_9)+\mathtt{E}\cdot(c'_{13}+k_{13})) \qquad (E_{b,2}) \\
\mathtt{2}\cdot b = {} & S^{-1}(\mathtt{D}\cdot(c_0+k_0)+\mathtt{9}\cdot(c_4+k_4)+\mathtt{9}\cdot(c_8+k_8)+\mathtt{D}\cdot(c_{12}+k_{12}))+ \\
& S^{-1}(\mathtt{D}\cdot(c'_0+k_0)+\mathtt{9}\cdot(c'_4+k_4)+\mathtt{9}\cdot(c'_8+k_8)+\mathtt{D}\cdot(c'_{12}+k_{12})) \qquad (E_{b,3})
\end{aligned}$$

The fault values a,b,c and d are unknown.

Key Filtering

The correct key will satisfy the fault equations derived above.In this attack, we identify large sets of key candidates which are inconsistent with some fault equations and exclude them from our keyspace. We reduce the keyspace to the extent when exhaustive search is computationally feasible. This can be done as follows:

- Key Tuple Filtering:
  - Every fault equation depends on 4 key indeterminates i.e. from k_0 to k_{15}.
  - The result of the fault equation must belong to Galois Field 16.
  - The correct key will satisfy all the 16 fault equations.
  - All such possible key intermediates are stored as 4 tuples.
- Key Set Filtering: Now we have the fault values of a,b,c and d. We can use this information to further reduce the keyspace by calculating which of these values regularly satisfy all 16 fault equations.

- Exhaustive Search: Now we have the set of all possible key values in the keyspace and now we will apply exhaustive search(brute force).

Finally, we were able to reduce our keyspace from 2^{64} to 2^{24}.

5 Conclusion

In this term paper, we discussed the design specifications of LED along with its security analysis and a thorough fault attack. It is apparent that its design and the non-existent key schedule in particular is very interesting. As mentioned by its designers, the cipher has not been much cryptanalysed independently. Therefore, it should not be deployed straightaway into devices. A major takeaway from this study has been the revival of the neglected field of key-schedule design. This block cipher might also act as a gateway for other researchers to come up with efficient ciphers in the design space for constrained hardware implementations due to its growing applications every day.

6 References

1. The LED Block Cipher
2. A Fault Attack on the LED Block Cipher
3. A. Moradi, A. Poschmann, S. Ling, C. Paar and H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In K. Paterson, editor, Advances in Cryptology - EUROCRYPT 2011, LNCS, Springer, 2011, to appear.
4. S. Park, S.H. Sung, S. Lee and J. Lim. Improving the Upper Bound on the Maximum Diferential and the Maximum Linear Hull Probability for SPN Structures and AES. In Thomas Johansson, editor, Fast Software Encryption - FSE 2003, volume 2887 of LNCS, pages 247-260. Springer, 2003.