Noname manuscript No.
(will be inserted by the editor)

An argument on the security of `LRBC`, a recently proposed lightweight block cipher

Sadegh Sadeghi · Nasour Bagheri

Received: date / Accepted: date

Abstract `LRBC` is a new lightweight block cipher that has been proposed for resource-constrained IoT devices. The cipher is claimed to be secure against differential cryptanalysis and linear cryptanalysis. However, beside short state length which is only 16-bits, the structures of the cipher only use the linear operations, the its s-boxes, and this is a reason why the cipher is completely insecure against the mentioned attacks. we present a few examples to show that. Also, we show that the round function of `LRBC` has some structural problem and even if we fix them the cipher does not provide complete diffusion. Hence, even with replacement of the cipher s-boxes with proper s-boxes, the problem will not be fixed and it is possible to provide deterministic distinguisher for any number of round of the cipher. In addition, we show that for any fixed key, it is possible to create a full code book for the cipher with the complexity of $2^{n/2}$, which should be compared with 2^n for any secure n-bit block cipher.

Keywords Differential Cryptanalysis · Linear Cryptanalysis · Full-code-book · `LRBC`

1 Introduction

Internet of Things (IoT) received a lot of attention during the last decade. In an IoT system, multiple objects interact and cooperate to provide different

---

S. Sadeghi
Department of Mathematics, Faculty of Mathematical Sciences and Computer, Kharazmi University, Tehran, Iran E-mail: s.sadeghi.khu@gmail.com

N. Bagheri
Electrical Engineering Department, Shahid Rajaee Teacher Training University, Tehran 16788-15811, Iran
and School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
E-mail: nbagheri@sru.ac.ir

services and provide accessibility at any time from many points. Examples of the important application of IoT are Internet of Vehicles (IoV), Internet of Energy (IoE), Internet of Sensors (IoS) and Machine to Machine Communications (M2M) [12]. It is expected the worldwide number of connected devices to increase to 125 billion connected devices by 2030, while it was nearly 27 billion connected devices in 2017 [19,20] with a global market to reach US $ 1,102.6 billion by 2026 [8].

However, advances in IoT architectures and protocols are still necessary to make the vision of the IoT reality. More notably, designing a secure protocol for many IoT applications is still a challenge, given the constrained devices in the edge, e.g. RFID tags. To provide desired security, it is not always possible to use common solution based on conventional cryptographic primitives, because those primitives such as AES [1] or SHA3 [22] do not meet the resource limitation of RFID tags. Hence, many lightweight primitives have been proposed last decade, targeting such applications. To just name some of such lightweight primitives, we can mention `SKINNY` [4], `PRESENT` [10], `MIBS` [17], `SIMON` [3], `SPECK` [3], `LS-Designs` [15], `ZORRO` [14] and `Fides` [7], `Quark` [2] and `PHOTON` [16]. In addition, recently NIST also initiated lightweight cryptography competition, targeting standardization of hash function and AEAD (authenticated encryption with associated data) for constrained environments which received 57 submissions for the first round and it is in the second round now [13].

In this direction, Biswas *et al.* recently proposed a lightweight block cipher called `LRBC` [9]. Designers of this block cipher have investigated its security against the well known attacks include linear and differential cryptanalysis [21, 6], impossible differential cryptanalysis [5,18], Zero-correlation linear cryptanalysis [11], and etc. The goal of differential and linear cryptanalysis is to find the high-probability features of the plaintexts propagate to the ciphertexts, called distinguisher. If the probability of a distinguisher in the target block cipher is obviously higher than that of a completely random permutation operation, that block cipher can be distinguished from a random permutation. Impossible differential attack is one of the most popular cryptanalytic tools for block ciphers. Impossible differential cryptanalysis starts with finding an input difference which results in an output difference with probability 0. Zero-correlation cryptanalysis is also a novel cryptanalytic approach, proposed by Bogdanov and Rijmen [11]. In contrast to conventional linear cryptanalysis which uses linear approximations with high correlation, zero-correlation linear cryptanalysis is based on linear approximations with a correlation exactly equal to zero for all keys.

`LRBC` is a lightweight block cipher proposed by Biswas *et al.* in 2020 [9]. The design takes both Feistel and SPN structure. The `LRBC` has been implemented using simple logical operations such as XOR operations ($\oplus$), XNOR operations ($\odot$), concatenation ($||$), transposition process. In this cipher, the long plaintext has been split into 16-bit blocks of data. In this paper, we analyze the security of this block cipher, which is its first third-party analysis to the best of our knowledge.

In the rest of the paper, in section 2 we describe LRBC briefly and also provide required preliminaries. In section 3 we provide our analysis of this cipher. Finally, the paper is concluded in section 4

2 Preliminaries

The encryption process of LRBC has been illustrated in Algorithm 1 and its F-Function is described in Algorithm 2. In these algorithms, $\mathcal{X}[i]$ defines i-th bit of string $\mathcal{X}$.

Algorithm 1 *LRBC Encryption [9]*

```
    Input: Plaintext (PT)

 1. Read plaintext (PT) and extract the byte values.
 2. $PT = PT_1 \| \ldots \| Pt_n$ and $PT_i \in \{0,1\}^{16}$, for $1 \leq i \leq n$.
 3. Initialize $r$ with value $1$.
 4. Each $PT_i$ is further su-divided into 4 equal length parts $PT_i^k, 1 \leq k \leq 4, 1 \leq i \leq n$ as,
        $PT_i^1 = PT_i[1] \ || \ PT_i[2] \ || \ PT_i[9] \ || \ PT_i[10]$
        $PT_i^2 = PT_i[3] \ || \ PT_i[4] \ || \ PT_i[11] \ || \ PT_i[12]$
        $PT_i^3 = PT_i[5] \ || \ PT_i[6] \ || \ PT_i[13] \ || \ PT_i[14]$
        $PT_i^4 = PT_i[7] \ || \ PT_i[8] \ || \ PT_i[15] \ || \ PT_i[16]$
 5. Compute intermediate round cipher blocks as ($a \neq b \neq c \neq d$),
        $IC_i^1 = PT_i^1 \odot K^a$
        $IC_i^2 = PT_i^2 \oplus K^b$
        $IC_i^3 = PT_i^3 \oplus K^c$
        $IC_i^4 = PT_i^4 \odot K^d$
 6. Generate F-Function as,
        $F_i^1 = F\_Function(IC_i^1, IC_i^3)$
        $F_i^2 = F\_Function(IC_i^2, IC_i^4)$
 7. Generate input for next round as,
        $PT_i^1 = F_i^1[5:8]; PT_i^2 = F_i^2[5:8]$
        $PT_i^3 = F_i^1[1:4]; PT_i^4 = F_i^2[1:4]$
        $r = r + 1$
 8. If ($r < 24$)
        Go to step 5.
 9. Else
        Go to step 10.
10. $ICT_i^k = PT_i^k, 1 \leq k \leq 4, 1 \leq i \leq n$.
11. Generate Final Cipher as,
        $CT = ICT_i^1 || ICT_i^2 || ICT_i^3 || ICT_i^4$.
```

Algorithm 2 *F-Function [9]*

```
    Input: Intermediate cipher blocks $IC_i^1, IC_i^2, IC_i^3, IC_i^4$.
    Output: 16-bit ciphertext.

 1. S-box computation,
```

```
        $IS_i^1 = IC_i^1 \odot IC_i^3$
        $IS_i^2 = IC_i^1 \oplus 1$
        $IS_i^3 = IC_i^2 \odot IC_i^4$
        $IS_i^4 = IC_i^2 \oplus 0$
2. P-box computation,
        $P_i^1 = IS_i^1[1]||IS_i^2[4]||IS_i^1[2]||IS_i^2[3]$
        $P_i^2 = IS_i^1[3]||IS_i^2[2]||IS_i^1[4]||IS_i^2[1]$
        $P_i^3 = IS_i^3[1]||IS_i^4[4]||IS_i^3[2]||IS_i^4[3]$
        $P_i^4 = IS_i^3[3]||IS_i^4[2]||IS_i^3[4]||IS_i^4[1]$
3. L-box computation,
        $T_i[1] = (P_i^1[1] \oplus P_i^2[4]); X_i[1] = (P_i^1[1] \odot 0)$
        $T_i[2] = (P_i^1[2] \odot P_i^2[3]); X_i[2] = (P_i^1[2] \oplus 1)$
        $T_i[3] = (P_i^1[3] \oplus P_i^2[2]); X_i[3] = (P_i^1[3] \odot 0)$
        $T_i[4] = (P_i^1[4] \odot P_i^2[1]); X_i[4] = (P_i^1[4] \oplus 1)$
        $T_i[5] = (P_i^3[1] \oplus P_i^4[4]); X_i[5] = (P_i^2[1] \odot 0)$
        $T_i[6] = (P_i^3[2] \odot P_i^4[3]); X_i[6] = (P_i^2[2] \oplus 1)$
        $T_i[7] = (P_i^3[3] \oplus P_i^4[2]); X_i[7] = (P_i^2[3] \odot 0)$
        $T_i[8] = (P_i^3[4] \odot P_i^4[1]); X_i[8] = (P_i^2[4] \oplus 1)$
    $L_i(1) = T_i[1]||X_i[4]||T_i[2]||X_i[3]||T_i[3]||X_i[2]||T_i[4]||X_i[1]$
    $L_i(2) = T_i[5]||X_i[8]||T_i[6]||X_i[7]||T_i[7]||X_i[6]||T_i[8]||X_i[5]$
    $z = L_i(1)||L_i(2)$
4. End.
```

The key schedule process of `LRBC` also can be presented as K^1, K^2, K^3, K^4 where $K^i \in \{0,1\}^4$, $i = 1, \cdots, 4$. For encryption/decryption process of 24 rounds of `LRBC`, 24 number of possible combinations of keys can be used in each round. The design of the key combinations has been shown in Table 1.

Table 1 The key combinations of all rounds of `LRBC` cipher as K^i, K^j, K^k, K^l.

Round	i	j	k	l	Round	i	j	k	l
1	1	2	3	4	13	3	2	1	4
2	1	2	4	3	14	3	2	4	1
3	1	3	2	4	15	3	1	2	4
4	1	3	4	2	16	3	1	4	2
5	1	4	3	2	17	3	4	1	2
6	1	4	2	3	18	3	4	2	1
7	2	1	3	4	19	4	2	1	3
8	2	1	4	3	20	4	2	3	1
9	2	3	1	4	21	4	3	2	1
10	2	3	4	1	22	4	3	1	2
11	2	4	3	1	23	4	1	3	2
12	2	4	1	3	24	4	1	2	3

3 Security analysis of `LRBC`

The designers of `LRBC` provided security analysis against differential and linear cryptanalysis [9]. According to their analysis, the `LRBC` is safe against these attacks. However, based on the structure of the `LRBC` algorithm, all the operations used in this algorithm are linear, therefore this is the reason that shows the `LRBC` is vulnerable against known attacks such as the differential, linear, impossible differential, zero-correlation attacks and also other attacks. In the following, we give a few examples to illustrate the vulnerability of the `LRBC` algorithm to the attacks mentioned above. Before that we prove the F-Function of `LRBC` cipher (see Algorithm 2) is not a permutation.

Remark 1 Based on the Algorithm 1, Step 6, F_i^1 and F_i^2 generates from (IC_i^1, IC_i^3) and (IC_i^2, IC_i^4), respectively. It shows F_i^1 and F_i^2 are independent. But according to Algorithm 2, $F_i^2 (= L_i(2))$ is dependent to $(IC_i^1, IC_i^2, IC_i^3, IC_i^4)$ [1] and so this shows that the F-Function of `LRBC` cipher can not be a permutation and we prove it in the following property.

Property 1 Let $F : \{0,1\}^{16} \rightarrow \{0,1\}^{16}$ is F-Function of `LRBC` cipher. For any $\mathtt{P} \in \{0,1\}^{16}$, and $\mathtt{M} \in \{0,1\}^4$, we have $F(\mathtt{P}) = F(\mathtt{P} \oplus \mathtt{0M00})$.

Proof For simplicity, in this proof, we use the same notation of Algorithm 2. We use the index $i = 1$, and $i = 2$ for the inputs $P_1 = \mathtt{P}$ and $P_2 = \mathtt{P} \oplus \mathtt{0M00}$, respectively and show $F(P_1) = F(P_2)$. Based on the notation of Algorithm 2, $P_1 = IC_1^1||IC_1^2||IC_1^3||IC_1^4$, and $P_2 = IC_2^1||IC_2^2||IC_2^3||IC_2^4 = IC_1^1||IC_1^2 \oplus \mathtt{M}||IC_1^3||IC_1^4$. Since, the only difference in P_1 and P_2 is in the second nible, so in the *S-box computation* phase the IS_2^1 and IS_2^2 for P_2 will remain unchanged and equal with IS_1^1 and IS_1^2, respectively. But the nibles IS_2^3 and IS_2^4 are changed as $IS_2^3 = IS_1^3 \oplus \mathtt{M}$, and $IS_2^4 = IS_1^4 \oplus \mathtt{M}$. In the *P-box computation* phase, only the P_2^3 and P_2^4 are affected by IS_2^3 and IS_2^4 and so we have $(\mathtt{M} = (m_1||m_2||m_3||m_4))$:

$$P_2^3 = IS_1^3[1] \oplus m_1||IS_1^4[4] \oplus m_4||IS_1^3[2] \oplus m_2||IS_1^4[3] \oplus m_3,$$
$$P_2^4 = IS_1^3[3] \oplus m_3||IS_1^4[2] \oplus m_2||IS_1^3[4] \oplus m_4||IS_1^4[1] \oplus m_1.$$

Since, in the *P-box computation* phase, the P_2^1 and P_2^2 did not change and are the same with P_1^1 and P_1^2, respectively, hence in the *L-box computation* phase, the $X_2[1]$ to $X_2[8]$ and also, $T_2[1]$ to $T_2[4]$ will remain unchange and only the $T_2[5]$ to $T_2[8]$ will change as

$$T_2[5] = (P_2^3[1] \oplus P_2^4[4]) = (IS_1^3[1] \oplus m_1 \oplus IS_1^4[1] \oplus m_1),$$
$$T_2[6] = (P_2^3[2] \odot P_2^4[3]) = (IS_1^4[4] \oplus m_4 \oplus IS_1^3[4] \oplus m_4),$$
$$T_2[7] = (P_2^3[3] \oplus P_2^4[2]) = (IS_1^3[2] \oplus m_2 \oplus IS_1^4[2] \oplus m_2),$$
$$T_2[8] = (P_2^3[4] \odot P_2^4[1]) = (IS_1^4[3] \oplus m_3 \oplus IS_1^3[3] \oplus m_3),$$

Based on the above equations, we have $T_2[5] = T_1[5]$, $T_2[6] = T_1[6]$, $T_2[7] = T_1[7]$, and $T_2[8] = T_1[8]$. Thus, $L_1(1)||L_1(2) = L_2(1)||L_2(2)$, and hence $F(P_1) = F(P_2)$.

---

[1] Hence, we have considered the step 6 of Algorithm 1 as $(F_i^1, F_i^2) = F\_Function(IC_i^1, IC_i^2, IC_i^3, IC_i^4)$.

Differential and Impossible Differential attack. Property 1 helps to creat differential characteristics with non-zero differential inputs to zero differential outputs with a probability of one for 24 rounds of `LRBC` algorithm. For a few examples, we can have the following characteristics ($\varDelta_{in}$ and $\varDelta_{out}$ shows the input and output differential, respectively).

$$\varDelta_{in} = \texttt{0001} \rightarrow \varDelta_{out} = \texttt{0000},$$
$$\varDelta_{in} = \texttt{0002} \rightarrow \varDelta_{out} = \texttt{0000},$$
$$\varDelta_{in} = \texttt{0003} \rightarrow \varDelta_{out} = \texttt{0000},$$
$$\varDelta_{in} = \texttt{0021} \rightarrow \varDelta_{out} = \texttt{0000},$$
$$\varDelta_{in} = \texttt{3133} \rightarrow \varDelta_{out} = \texttt{0000},$$

and two examples in case of non-zero input to non-zero output are as follows:

$$\varDelta_{in} = \texttt{0009} \rightarrow \varDelta_{out} = \texttt{b525},$$
$$\varDelta_{in} = \texttt{d3fb} \rightarrow \varDelta_{out} = \texttt{4968}.$$

Obviously, any differential characteristic that have the probability of one can lead to many impossible differential characteristic. For example, all differential characteristic as $\varDelta_{in} = \texttt{0001} \rightarrow (\varDelta_{out} \neq 0) \in \{0,1\}^4$ are impossible differential characteristics for 24 rounds of `LRBC` and so on.

Linear and Zero correlation attack. We could not find a linear characteristic with the probability except $\frac{1}{2}$ and so all characteristics that we searched have a bias equal to 0. Therefore, these characteristics can lead to a zero correlation attack. The following is a few examples of this type of characteristics.

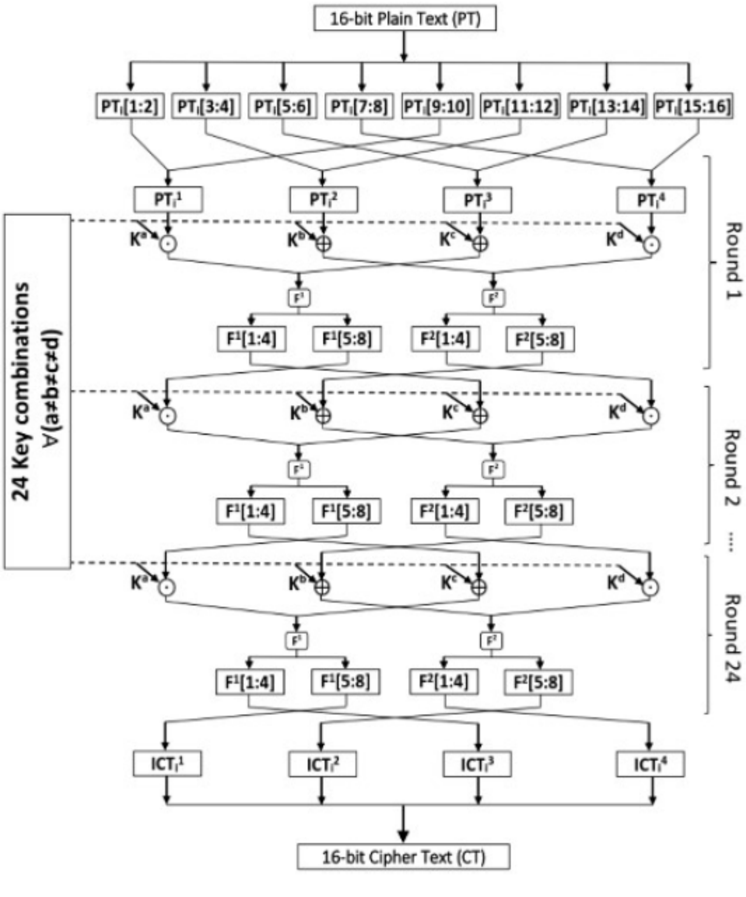
$$\varGamma_{in} = \texttt{0002} \rightarrow \varGamma_{out} = \texttt{1000},$$
$$\varGamma_{in} = \texttt{105b} \rightarrow \varGamma_{out} = \texttt{16ec},$$
$$\varGamma_{in} = \texttt{24a1} \rightarrow \varGamma_{out} = \texttt{000f},$$

where $\varGamma_{in}$ and $\varGamma_{out}$ shows the input and output linear masks, respectively.

3.1 A discussion on `LRBC` structure

According to our analysis above, the design of this algorithm has obvious bugs. One of the most important drawbacks besides being linear is having a non-permutation function in its structure that this is due to the use of depended functions F^1 and F^2. But, the designers also presented the graphical representation of encryption process of `LRBC` as shown in Fig. 1 (*we borrowed this image from the original paper [9] intentionally*). Based on this graphical representation, the F^1 and F^2 functions must be independent of each other. Hence, it shows there should be some typos in the Alg 2 of designers. In fact we guess the P_i^2 that is used to generate $X_i[5]$ to $X_i[8]$ in the *L-box computation* phase of Algorithm 2, should be replace by P_i^3. Thus, $X_i[5]$ to $X_i[8]$ will be as $X_i[5] = (P_i^3[1] \odot 0)$, $X_i[6] = (P_i^3[2] \oplus 1)$, $X_i[7] = (P_i^3[3] \odot 0)$,

Fig. 1 Graphical representation of encryption process of LRBC [9]



and $X_i[8] = (P_i^3[4] \oplus 1)$. By applying these changes, the F-Function of LRBC cipher will be a permutation and the details of Algorithm 2 can be the same as the graphical representation shown in Fig. 1.

Note that although correcting these typos causes to F-Function of LRBC be a permutation, the LRBC cipher remains insecure against the attacks mentioned above due to linearity of all operations that are used in the cipher. However, in the following we show that even by considering a nonlinear operation in the LRBC's F-Function, the structure of cipher will not have the necessary safety. The claim comes from that half the encrypted plaintext is encrypted independently of the other half. As it can be seen in the Fig. 1, the path that passes through the F^1 function is completely independent of the path that the F^2 function uses. Therefore, the time complexity of creating a code-book for LRBC is only $2^8 = 256$ instead of 2^{16}. Hence, we can create a full code-book only by query 256 chosen-ciphertext. For more details, it is enough to choose 256 chosen-ciphertext as $CT = ICT_i^1||ICT_i^2||ICT_i^3||ICT_i^4 = *|| * || \diamond$

$||\diamond$ to obtain 256 corresponding plaintext $P_{*\diamond}$ with a fixed key, where $*, \diamond \in \{\mathtt{0}, \mathtt{1}, \cdots, \mathtt{f}\}$. Now, for a given ciphertext as $CT = k||l||m||n$, the plaintext will be as $\big(< P_{km}.\mathtt{f0f0} > \oplus < P_{ln}.\mathtt{0f0f} > \big)$, where $< .,. >$ shows the inner product.

4 Conclusion

In this work, we analyzed the security of `LRBC` block cipher and showed that the design of this cipher have some structural problems and since it does not use nonlinear operators, so it is insecure against the known attacks.It should be noted the message/key length in this cipher is only 16- bits. Hence even doing exhaustive search only costs 2^{16}. However, our analysis shows that the cipher insecurity is structural and for example one can not fix it by using changing the word length from 4 to 16 and replacing the 4-bit s-boxes by 16-bit perfect s-boxes. Even in that case the complexity of creating a full-code-book for the cipher will be 2^{32} not 2^{64}. This study once again highlight the important of proper security analysis of any new primitive to avoid trivial attacks.

It should be noted, the designers have not made their reference-implementations publicly available. Hence, we put our implementation available at the end of this paper for any possible use. In addition, we have an implementation available at this link: http://cpp.sh/6reup

References

1. AES: AES: the Advanced Encryption Standard (1997). `http://competitions.cr.yp.to/aes.html`
2. Aumasson, J., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. J. Cryptology **26**(2), 313–339 (2013). DOI 10.1007/s00145-012-9125-6. URL `https://doi.org/10.1007/s00145-012-9125-6`
3. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK lightweight block ciphers. In: Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015, pp. 175:1–175:6. ACM (2015)
4. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: M. Robshaw, J. Katz (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II, *Lecture Notes in Computer Science*, vol. 9815, pp. 123–153. Springer (2016)
5. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp. 12–23. Springer (1999)
6. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. Journal of CRYPTOLOGY **4**(1), 3–72 (1991)
7. Bilgin, B., Bogdanov, A., Knezevic, M., Mendel, F., Wang, Q.: Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In: G. Bertoni, J. Coron (eds.) Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 8086, pp. 142–158. Springer (2013)

8. BIS: Internet of things market analysis 2026. `https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307` (2019 - Last accessed on 23 march 2020)
9. Biswas, A., Majumdar, A., Nath, S., Dutta, A., Baishnab, K.: Lrbc: a lightweight block cipher design for resource constrained iot devices. Journal of Ambient Intelligence and Humanized Computing pp. 1–15 (2020)
10. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: P. Paillier, I. Verbauwhede (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings, *Lecture Notes in Computer Science*, vol. 4727, pp. 450–466. Springer (2007)
11. Bogdanov, A., Rijmen, V.: Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Designs, codes and cryptography **70**(3), 369–383 (2014)
12. Ferrag, M.A., Maglaras, L.A., Janicke, H., Jiang, J., Shu, L.: Authentication protocols for internet of things: A comprehensive survey. Security and Communication Networks **2017**, 6562953:1–6562953:41 (2017). DOI 10.1155/2017/6562953. URL `https://doi.org/10.1155/2017/6562953`
13. fgs: Nist lightweight cryptography standardization process. In: adsgad, pp. 2–3. Springer (accessed 01 Novamber 2019). URL `https://csrc.nist.gov/projects/lightweight-cryptography`
14. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.: Block ciphers that are easier to mask: How far can we go? In: G. Bertoni, J. Coron (eds.) Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 8086, pp. 383–399. Springer (2013)
15. Grosso, V., Leurent, G., Standaert, F., Varici, K.: Ls-designs: Bitslice encryption for efficient masked software implementations. In: C. Cid, C. Rechberger (eds.) Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 8540, pp. 18–37. Springer (2014)
16. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: P. Rogaway (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings, *Lecture Notes in Computer Science*, vol. 6841, pp. 222–239. Springer (2011)
17. Izadi, M., Sadeghiyan, B., Sadeghian, S.S., Khanooki, H.A.: MIBS: A new lightweight block cipher. In: J.A. Garay, A. Miyaji, A. Otsuka (eds.) Cryptology and Network Security, 8th International Conference, CANS 2009, Kanazawa, Japan, December 12-14, 2009. Proceedings, *Lecture Notes in Computer Science*, vol. 5888, pp. 334–348. Springer (2009)
18. Knudsen, L.: Deal-a 128-bit block cipher. complexity **258**(2), 216 (1998)
19. Markit, I.: Number of connected iot devices will surge to 125 billion by 2030. `https://technology.informa.com/596542` (2017- Last accessed on 29 March 2020)
20. Markit, I.: The internet of things: a movement, not a market. Englewood, CO: IHS Markit. `https://cdn.ihs.com/www/pdf/IoT_ebook.pdf` **28**, 2018 (2017, Last accessed on 23 march 2020)
21. Matsui, M.: Linear cryptanalysis method for des cipher. In: Workshop on the Theory and Application of of Cryptographic Techniques, pp. 386–397. Springer (1993)
22. SHA3: SHA-3: a Secure Hash Algorithm (2007). `http://competitions.cr.yp.to/sha3.html`

A C++ source code for encryption process of LRBC block cipher

```
// Encryption process of LRBC block cipher
#include<iostream>
#include <bitset>
```

```
using namespace std;
// the number of rounds.
#define ROUNDS (24)

// The F-function based on the Alg 2. Page 6 in the LRBC paper.
void F_Function(int round, int IC1[][4], int IC2[][4], int IC3[][4],
int IC4[][4], int F1[][8], int F2[][8]);

// Structure of LRBC keys based on Fig. 2 Page 5 in the LRBC paper.
void Key_schedule(int key, int key_a[][4], int key_b[][4],
int key_c[][4], int key_d[][4]);
// Encryption process function
int Encryption_Process(int palintext, int key);
#define Xnor(a , b) (a ^ b ^ 1) // Ex-NOR function
#define Xor(a,b) (a ^ b) // Ex-OR function

int main() {
// read 16-bit PLAINTEXT and KEY
        int palintext = 0x0021;
        int key = 0x234f;
        int ciphertext = { 0 };
        ciphertext = Encryption_Process(palintext, key);
// Print Plaintext
                std::cout << ”Plaintext:\t”;
                std::cout << hex << palintext;
                std::cout << ”\n”;
// Print key
                std::cout << ”Key:\t\t”;
                std::cout << hex << key;
                std::cout << ”\n”;
 // Print ciphertext
                std::cout << ”Ciphertext:\t”;
                std::cout << hex << ciphertext;
                std::cout << ”\n”;
        return 0;
}
// F-function based on the Alg 2. of Page 6 in the LRBC paper.
void F_Function(int round, int IC1[][4], int IC2[][4], int IC3[][4],
int IC4[][4],int L1[][8], int L2[][8]) {
//S-box computation
        int IS1[4] = { 0 };
        int IS2[4] = { 0 };
        int IS3[4] = { 0 };
        int IS4[4] = { 0 };

        for (int j = 0; j < 4; j++) {
                IS1[j] = Xnor(IC1[round - 1][j], IC3[round - 1][j]);

                if (j != 3)
                        IS2[j] = IC1[round - 1][j];
                else
                        IS2[j] = Xor(IC1[round - 1][j], 1);

                IS3[j] = Xnor(IC2[round - 1][j], IC4[round - 1][j]);
                IS4[j] = IC2[round - 1][j];
        }
// P-box computation
        int P1[4] = { 0 };
```

```
        int P2[4] = { 0 };
        int P3[4] = { 0 };
        int P4[4] = { 0 };
        P1[0] = IS1[0];
        P1[1] = IS2[3];
        P1[2] = IS1[1];
        P1[3] = IS2[2];
        P2[0] = IS1[2];
        P2[1] = IS2[1];
        P2[2] = IS1[3];
        P2[3] = IS2[0];
        P3[0] = IS3[0];
        P3[1] = IS4[3];
        P3[2] = IS3[1];
        P3[3] = IS4[2];
        P4[0] = IS3[2];
        P4[1] = IS4[1];
        P4[2] = IS3[3];
        P4[3] = IS4[0];
// l-box computation
        int T[8] = { 0 };
        int X[8] = { 0 };
        T[0] = Xor(P1[0], P2[3]);
        T[1] = Xnor(P1[1], P2[2]);
        T[2] = Xor(P1[2], P2[1]);
        T[3] = Xnor(P1[3], P2[0]);
        T[4] = Xor(P3[0], P4[3]);
        T[5] = Xnor(P3[1], P4[2]);
        T[6] = Xor(P3[2], P4[1]);
        T[7] = Xnor(P3[3], P4[0]);
        X[0] = Xnor(P1[0], 0);
        X[1] = Xor(P1[1], 1);
        X[2] = Xnor(P1[2], 0);
        X[3] = Xor(P1[3], 1);
        X[4] = Xnor(P2[0], 0);
        X[5] = Xor(P2[1], 1);
        X[6] = Xnor(P2[2], 0);
        X[7] = Xor(P2[3], 1);
// Output --> L1[][] is L(1) and L2[][] is L(2) in in the LRBC paper.
        L1[round - 1][0] = T[0];
        L1[round - 1][1] = X[3];
        L1[round - 1][2] = T[1];
        L1[round - 1][3] = X[2];
        L1[round - 1][4] = T[2];
        L1[round - 1][5] = X[1];
        L1[round - 1][6] = T[3];
        L1[round - 1][7] = X[0];
        L2[round - 1][0] = T[4];
        L2[round - 1][1] = X[7];
        L2[round - 1][2] = T[5];
        L2[round - 1][3] = X[6];
        L2[round - 1][4] = T[6];
        L2[round - 1][5] = X[5];
        L2[round - 1][6] = T[7];
        L2[round - 1][7] = X[4];
}
/* Structure of LRBC key based on the Fig. 2 of Page 5
in the LRBC paper.*/
```

```
void Key_schedule(int key, int key_a[][4], int key_b[][4],
int key_c[][4], int key_d[][4]) {
        int K[16];
        for (int j = 0; j < 16; j++) {
                K[(15 - j)] = bitset<16>(key)[j];
        }
        int k1[4], k2[4], k3[4], k4[4];
        for (int j = 0; j < 16; j++) {
                if (j < 4)
                        k1[j] = K[j];
                else if (4 <= j && j < 8)
                        k2[j - 4] = K[j];
                else if (8 <= j && j < 12)
                        k3[j - 8] = K[j];
                else if (12 <= j && j < 16)
                        k4[j - 12] = K[j];
        }
        for (int j = 0; j < 4; j++) {
                key_a[0][j] = k1[j];
                key_b[0][j] = k2[j];
                key_c[0][j] = k3[j];
                key_d[0][j] = k4[j]; // round 1
                key_a[1][j] = k1[j];
                key_b[1][j] = k2[j];
                key_c[1][j] = k4[j];
                key_d[1][j] = k3[j]; // round 2
                key_a[2][j] = k1[j];
                key_b[2][j] = k3[j];
                key_c[2][j] = k2[j];
                key_d[2][j] = k4[j]; // round 3
                key_a[3][j] = k1[j];
                key_b[3][j] = k3[j];
                key_c[3][j] = k4[j];
                key_d[3][j] = k2[j]; // round 4
                key_a[4][j] = k1[j];
                key_b[4][j] = k4[j];
                key_c[4][j] = k3[j];
                key_d[4][j] = k2[j]; // round 5
                key_a[5][j] = k1[j];
                key_b[5][j] = k4[j];
                key_c[5][j] = k2[j];
                key_d[5][j] = k3[j]; // round 6
                key_a[6][j] = k2[j];
                key_b[6][j] = k1[j];
                key_c[6][j] = k3[j];
                key_d[6][j] = k4[j]; // round 7
                key_a[7][j] = k2[j];
                key_b[7][j] = k1[j];
                key_c[7][j] = k4[j];
                key_d[7][j] = k3[j]; // round 8
                key_a[8][j] = k2[j];
                key_b[8][j] = k3[j];
                key_c[8][j] = k1[j];
                key_d[8][j] = k4[j]; // round 9
                key_a[9][j] = k2[j];
                key_b[9][j] = k3[j];
                key_c[9][j] = k4[j];
                key_d[9][j] = k1[j]; // round 10
```

```
        key_a[10][j] = k2[j];
        key_b[10][j] = k4[j];
        key_c[10][j] = k3[j];
        key_d[10][j] = k1[j]; // round 11
        key_a[11][j] = k2[j];
        key_b[11][j] = k4[j];
        key_c[11][j] = k1[j];
        key_d[11][j] = k3[j]; // round 12
        key_a[12][j] = k3[j];
        key_b[12][j] = k2[j];
        key_c[12][j] = k1[j];
        key_d[12][j] = k4[j]; // round 13
        key_a[13][j] = k3[j];
        key_b[13][j] = k2[j];
        key_c[13][j] = k4[j];
        key_d[13][j] = k1[j]; // round 14
        key_a[14][j] = k3[j];
        key_b[14][j] = k1[j];
        key_c[14][j] = k2[j];
        key_d[14][j] = k4[j]; // round 15
        key_a[15][j] = k3[j];
        key_b[15][j] = k1[j];
        key_c[15][j] = k4[j];
        key_d[15][j] = k2[j]; // round 16
        key_a[16][j] = k3[j];
        key_b[16][j] = k4[j];
        key_c[16][j] = k1[j];
        key_d[16][j] = k2[j]; // round 17
        key_a[17][j] = k3[j];
        key_b[17][j] = k4[j];
        key_c[17][j] = k2[j];
        key_d[17][j] = k1[j]; // round 18
        key_a[18][j] = k4[j];
        key_b[18][j] = k2[j];
        key_c[18][j] = k1[j];
        key_d[18][j] = k3[j]; // round 19
        key_a[19][j] = k4[j];
        key_b[19][j] = k2[j];
        key_c[19][j] = k3[j];
        key_d[19][j] = k1[j]; // round 20
        key_a[20][j] = k4[j];
        key_b[20][j] = k3[j];
        key_c[20][j] = k2[j];
        key_d[20][j] = k1[j]; // round 21
        key_a[21][j] = k4[j];
        key_b[21][j] = k3[j];
        key_c[21][j] = k1[j];
        key_d[21][j] = k2[j]; // round 22
        key_a[22][j] = k4[j];
        key_b[22][j] = k1[j];
        key_c[22][j] = k3[j];
        key_d[22][j] = k2[j]; // round 23
        key_a[23][j] = k4[j];
        key_b[23][j] = k1[j];
        key_c[23][j] = k2[j];
        key_d[23][j] = k3[j]; // round 24
    }
}
```

```
int Encryption_Process(int palintext, int key)
{
        int START_ROUNDS(0);
// Converting plaintext to the PT as array
        int PT[16] = { 0 };
        for (int j = 0; j < 16; j++) {
                PT[(15 - j)] = bitset<16>(palintext)[j];
        }
// Definr Variables
        int PT1[ROUNDS + 1][4]= { 0 };
        int PT2[ROUNDS + 1][4]= { 0 };
        int PT3[ROUNDS + 1][4]= { 0 };
        int PT4[ROUNDS + 1][4]= { 0 };
        int IC1[ROUNDS][4]= { 0 };
        int IC2[ROUNDS][4]= { 0 };
        int IC3[ROUNDS][4]= { 0 };
        int IC4[ROUNDS][4]= { 0 };
        int F1[ROUNDS][8]= { 0 };
        int F2[ROUNDS][8]= { 0 };
        int key_a[24][4] = { 0 };
        int key_b[24][4] = { 0 };
        int key_c[24][4] = { 0 };
        int key_d[24][4] = { 0 };
// Define the Key_schedule function
        Key_schedule(key, key_a, key_b, key_c, key_d);
/*Converting PT to the PTi (i=1,2,3,4) based on Step 4
of the Alg 1. in page 6 in the LRBC paper*/
        PT1[START_ROUNDS][0] = PT[0];
        PT1[START_ROUNDS][1] = PT[1];
        PT1[START_ROUNDS][2] = PT[8];
        PT1[START_ROUNDS][3] = PT[9];
        PT2[START_ROUNDS][0] = PT[2];
        PT2[START_ROUNDS][1] = PT[3];
        PT2[START_ROUNDS][2] = PT[10];
        PT2[START_ROUNDS][3] = PT[11];
        PT3[START_ROUNDS][0] = PT[4];
        PT3[START_ROUNDS][1] = PT[5];
        PT3[START_ROUNDS][2] = PT[12];
        PT3[START_ROUNDS][3] = PT[13];
        PT4[START_ROUNDS][0] = PT[6];
        PT4[START_ROUNDS][1] = PT[7];
        PT4[START_ROUNDS][2] = PT[14];
        PT4[START_ROUNDS][3] = PT[15];
// start rounds
        for (int r = 1; r <= ROUNDS; r++) {
//  Step 5 of Alg 1. in page 6 in the LRBC paper
                IC1[r - 1][0] = Xnor(PT1[r-1][0], key_a[r - 1][0]);
                IC1[r - 1][1] = Xnor(PT1[r-1][1], key_a[r - 1][1]);
                IC1[r - 1][2] = Xnor(PT1[r-1][2], key_a[r - 1][2]);
                IC1[r - 1][3] = Xnor(PT1[r-1][3], key_a[r - 1][3]);
                IC2[r - 1][0] = Xor(PT2[r-1][0], key_b[r - 1][0]);
                IC2[r - 1][1] = Xor(PT2[r-1][1], key_b[r - 1][1]);
                IC2[r - 1][2] = Xor(PT2[r-1][2], key_b[r - 1][2]);
                IC2[r - 1][3] = Xor(PT2[r-1][3], key_b[r - 1][3]);
                IC3[r - 1][0] = Xor(PT3[r-1][0], key_c[r - 1][0]);
                IC3[r - 1][1] = Xor(PT3[r-1][1], key_c[r - 1][1]);
                IC3[r - 1][2] = Xor(PT3[r-1][2], key_c[r - 1][2]);
                IC3[r - 1][3] = Xor(PT3[r-1][3], key_c[r - 1][3]);
```

```
                IC4[r - 1][0] = Xnor(PT4[r-1][0], key_d[r - 1][0]);
                IC4[r - 1][1] = Xnor(PT4[r-1][1], key_d[r - 1][1]);
                IC4[r - 1][2] = Xnor(PT4[r-1][2], key_d[r - 1][2]);
                IC4[r - 1][3] = Xnor(PT4[r-1][3], key_d[r - 1][3]);
// Define F-function ( Step 6 of the Alg 1. in page 6 in the LRBC paper)
                F_Function(r, IC1, IC2, IC3, IC4, F1, F2);
// Step 7 of the Alg 1. in page 6 in the LRBC paper
                for (int j = 0; j < 4; j++) {
                        PT1[r][j] = F1[r - 1][j + 4];
                        PT2[r][j] = F2[r - 1][j + 4];
                        PT3[r][j] = F1[r - 1][j];
                        PT4[r][j] = F2[r - 1][j];
                }
        }
//  Step 10 of the Alg 1. in page 6 in the LRBC paper
        int ICT[16] = { 0 };
        for (int j = 0; j < 4; j++) {
                ICT[j] = PT1[ROUNDS][j];
                ICT[j + 4] = PT2[ROUNDS][j];
                ICT[j + 8] = PT3[ROUNDS][j];
                ICT[j + 12] = PT4[ROUNDS][j];
        }
/* Converting ICT array to Ciphertext as Hex format
    and return Ciphertext*/
        int ciphertext = 0;
        for (int i = 0; i < 16; i++)
                if (ICT[i]) ciphertext |= (1 << (15 - i));
        return ciphertext;
}
```