Four Ciphers in One: Full-Round Attacks on the Reference Implementation of LLBC

Sei Kofuku and Yasutaka Igarashi

Department of Electrical Engineering, Tokyo University of Science, Noda, Chiba, Japan

Abstract

LLBC is a 128-bit low-latency block cipher for IoT applications, proposed in the IEEE Internet of Things Journal in 2025. The designers release a reference implementation in C and Verilog for reproducibility. The test vectors in the proposal are outputs of this implementation. We show that the reference implementation differs from the specification of the peer-reviewed proposal in eight points. In particular, its round function uses the rotation amounts $(32, 16)$ instead of $(3, 19)$. We prove that an LLBC-type cipher with rotation amounts (b_1, b_2) is a product of $g/4$ independent ciphers when $g = \gcd(64, b_1, b_2)$ is a multiple of 4. Hence the cipher splits when g is a multiple of 8. As a result, the reference implementation splits into four 32-bit ciphers that never mix. With this split, we attack all 20 rounds. We give a distinguisher with probability 1 from two chosen plaintexts. We decrypt any ciphertext with 2^{32} chosen plaintexts. We recover the master key from three known plaintexts with $2^{80} + 3 \cdot 2^{16}$ encryptions, which is below exhaustive search by a factor of 2^{48}. The designers estimated that the best attack on the specification reaches nine rounds. On a single PC, our decryption attack on all 20 rounds of the reference implementation took 175.5 seconds. We also study standard techniques. A standard automatic search returns 20-round impossible differentials and zero-correlation approximations. In contrast, the classical differential and linear characteristics and the integral properties that we examined distinguish only 9, 10, and 8 rounds, respectively, inside one part. We also show that the assumption of the usual models, which treat the S-boxes as independent, does not hold for the reference implementation. Finally, we recommend that designers check the released code against test vectors made independently from the specification and check the diffusion of the released code itself before publication.

Keywords: LLBC, reference implementation, lightweight block cipher, full-round attack, rotation amounts

1 Introduction

Background. Many low-latency and lightweight block ciphers have been proposed in recent years. They target applications that need short response times, such as IoT devices and memory encryption. Such a cipher is often published in two forms. One is a paper that gives the specification. The other is a reference implementation with test vectors, released by the designers. Users and third-party evaluators do not only read the specification. They also run the released implementation and check their own implementations with the test vectors. Therefore, the agreement between the released implementation and the specification is part of the basis for the security of a cipher. Third-party analysis has broken some lightweight ciphers proposed in the IEEE Internet of Things Journal in full

Emails: 7325550@ed.tus.ac.jp (Sei Kofuku), yasutaka@rs.noda.tus.ac.jp (Yasutaka Igarashi)

Date of this document: 2026-11-15.

Table 1: Attacks on the full 20-round reference implementation of LLBC-128-128. CP: chosen plaintexts. KP: known plaintexts. Time is counted in 20-round encryptions.

Attack	Data	Time	Memory	Experiment	Section
Distinguisher	2 CP	2	negligible	—	4.1
Decryption of any ciphertext	2^{32} CP	2^{32}	2^{39} bits	full scale	4.2
Master-key recovery	3 KP	$2^{80} + 3 \cdot 2^{16}$	negligible	reduced version	4.3
(Reference) exhaustive search	1–2 KP	2^{128}	negligible		

rounds. For example, Kim et al. [KSK+25] gave a distinguisher for Shadow [GLL21] that uses two plaintext–ciphertext pairs and works for any number of rounds.

LLBC is a Feistel-type low-latency block cipher proposed by Li et al. in the IEEE Internet of Things Journal in 2025 [LWP+25]. We call this paper *the proposal*. LLBC has a 128-bit block. LLBC-128-128 has a 128-bit key and 20 rounds. The designers evaluated its security with automatic search and other methods. They estimated that the best attack is a 9-round integral attack, and they set the number of rounds to $20 = 2 \times 9 + 2$. In the appendix of the proposal, they state that all code and test vectors are available on GitHub for reproducibility [LWP+25, Appendix].

Our results. In this paper, we analyze the reference implementation of LLBC-128-128 released by the designers, and we show that it is broken in all 20 rounds in practical time. The reference implementation uses the rotation amounts $(32, 16)$ in the round function. The specification of the peer-reviewed proposal uses $(3, 19)$. We prove that an LLBC-type cipher with rotation amounts (b_1, b_2) is a product of $g/4$ independent ciphers when $g = \gcd(64, b_1, b_2)$ is a multiple of 4. Hence the cipher splits when g is a multiple of 8. This split depends only on the rotation amounts. It does not depend on the S-boxes, the coefficients of the linear layer, the round keys, or the round constants.

For $(32, 16)$, we have $g = 16$. The 128-bit state splits into four groups of 32 bits that never mix. We call each group a *part*. In other words, the reference implementation is four 32-bit ciphers placed side by side, and each of them is an LLBC-type cipher with 16-bit halves (Figure 4). The designers estimated that the best attack on the specification reaches nine rounds. On a single PC, however, our attack decrypted ciphertexts of the full 20-round reference implementation in 175.5 seconds. **The main novelty of this paper is that we found and proved this split, which comes from the rotation amounts of the reference implementation, and that we broke the 128-bit cipher in full rounds as four 32-bit ciphers.** Table 1 summarizes the attacks.

Our contributions. Our three main contributions are as follows.

- **The split theorem (Section 3).** We prove the condition under which an LLBC-type cipher splits. The condition includes the cell boundaries of the S-boxes. We checked all 3969 pairs with $1 \le b_1, b_2 \le 63$. For every pair, the existence and the number of parts agree with the condition.
- **Attacks on all 20 rounds (Section 4).** We give the three attacks of Table 1: a distinguisher, the decryption of any ciphertext, and master-key recovery. We ran the decryption at full scale, and it recovered the plaintexts of all 1000 random ciphertexts. We ran the key recovery with fewer unknown dimensions, and it recovered the 128-bit master key.
- **Standard techniques on the reference implementation (Section 5).** A standard automatic search (STP) returns 20-round impossible differentials and zero-

correlation approximations for the whole 128-bit cipher. The split explains why. In contrast, the classical differential and linear characteristics and the integral properties that we examined distinguish only 9, 10, and 8 rounds, respectively, inside one part. We show this with minimum weights and with measurements over all 2^{32} inputs. These numbers are observations, not upper bounds for these techniques. Moreover, in the reference implementation, S_1 and S_2 read the same nibble without a key. Hence the assumption of the usual models, which treat the S-boxes as independent, does not hold.

In addition, we list the eight differences between the designers' release and the specification of the proposal (Section 2).

Target and scope. Our target is the reference implementation released by the designers themselves. A SHA-1 hash and a third-party archive identify the release (Section 2). The test vectors in the proposal (Table XVI of [LWP+25]) agree with this implementation, not with the specification. Thus a user who follows the released code and checks the result with these test vectors implements the cipher that we break. The rotation amounts $(3, 19)$ of the specification give $g = 1$, and no split occurs. We study differential attacks on the specification in a separate paper [KI26]. The specification is out of the scope of this paper.

CryptanalysisBench classifies breaks of ciphers [FSC+26, Section 4.2]. It treats a bug in a reference implementation that contradicts a clear specification as a separate class from weaknesses of the design itself. Our result is close to this class. However, we believe that our result goes beyond a report of an implementation bug, for three reasons. First, the target is the designers' own release, not a third-party implementation, and the test vectors of the proposal confirm it. Second, we give the condition for the split as a general theorem that depends only on the rotation amounts. Third, we show that standard automatic evaluation does not reveal that the assumptions of its models fail, and that it presents useless light characteristics as full-round distinguishers. This shows that the released code itself needs evaluation.

Related work. Kölbl et al. pointed out that the SIMON-like round function $x \odot S^d(x)$ partitions its output bits into independent classes when $\gcd(d, n) \neq 1$ [KLT15, Section 3.2]. Our work differs from theirs in two points. First, we decompose the whole cipher, not the round function. Second, our condition $(4 \mid g)$ includes the cell boundaries of the S-boxes, and we use it for full-round attacks. Many attacks on KLEIN use the fact that its high and low nibbles are almost independent [LNP15, Section 3]. In KLEIN, however, MixNibbles links the high and low nibbles. In contrast, the reference implementation of LLBC splits into fully independent parts in all rounds.

Disclosure and code. We informed the designers of these results on the day we submitted this paper to the IACR ePrint Archive. Our code for the experiments is available at `https://anonymous.4open.science/r/LLBC-3216-4D95/`. File names in typewriter font, such as `split_condition.py`, refer to scripts in this repository.

Organization. Section 2 describes the designers' release and its differences from the specification. Section 3 proves the split theorem. Section 4 gives the full-round attacks and the experiments. Section 5 studies how the reference implementation looks under standard techniques. Section 6 discusses the results, and Section 7 concludes the paper.

2 Target: The Reference Implementation of LLBC

2.1 The Designers' Release

Along with the proposal [LWP+25], the designers release an implementation in the GitHub repository `13est-Wishes/LLBC-cipher` [Git25]. The release is a single RAR archive, `LLBC_cipher.rar`. Commit `f30143d` added it on June 22, 2025. The archive that we analyzed has the following SHA-1 hash:

`a9f914fadd3027b4c1232789ab81e1a194ae02f6`

So that third parties can check the same archive, we saved it in the Wayback Machine of the Internet Archive on October 26, 2026, at

`https://web.archive.org/web/20261026114738/`
`https://raw.githubusercontent.com/13est-Wishes/LLBC-cipher/main/LLBC_cipher.rar`

The archive contains the following files.

- `C/LLBC.cpp`: a C implementation of encryption, decryption, and the key schedule, with a main function for testing. We call this code *the reference implementation*.
- `Hardware/`: the round function (`LLBCRound.v`) and the whole cipher (`LLBC.v`) in Verilog.
- `Software/opt32`: an optimized implementation for 32-bit platforms. Headers such as `cipher.h` are missing, and some functions are undefined. It does not compile as it is.

We translated the logic of the Verilog code into Python and compared it with the C implementation. The outputs agreed for all 2000 random pairs of plaintext and key. The port names of `LLBC.v` do not match those of `LLBCRound.v`, so the Verilog code cannot be synthesized as it is. Thus the two implementations whose logic is complete (C and Verilog) define the same cipher.

2.2 Notation

For a 64-bit word x, we write its bits as $x[0], \ldots, x[63]$, where $x[i]$ has weight 2^i. We write $x \lll r$ for the left rotation by r bits. The 4-bit S-boxes act on cells. Cell c ($c = 0, \ldots, 15$) consists of bits $4c, \ldots, 4c+3$. We write $S(x)$ for the word obtained by applying S to all 16 cells of x.

2.3 Encryption of the Reference Implementation

Figure 1 shows the round function of the specification of the proposal and that of the reference implementation side by side.

The state is a pair (L, R) of 64-bit words. For the four 32-bit words (x_0, x_1, x_2, x_3) of the C implementation, we have $L = x_0 \| x_1$ and $R = x_2 \| x_3$. Let (A_j, B_j) be the round key of round j ($j = 0, \ldots, 19$), a pair of 64-bit words. Let $RC = \texttt{0x243F6A88} \,\|\, \texttt{0x00000000}$. With rotation amounts (b_1, b_2), one round is as follows:

$$y_1 = S_1(L \lll b_1), \qquad y_2 = S_2(L \lll b_2), \tag{1}$$

$$L' = \mathrm{MW}_{b_1,b_2}(y_1, y_2) \oplus R \oplus B_j, \qquad R' = L \oplus A_j \oplus RC, \tag{2}$$

where MixWord (Figure 2) is

$$\begin{gathered} t = y_1 \oplus y_2, \qquad m_0 = (t \lll b_1) \oplus y_1, \qquad m_1 = (m_0 \lll b_2) \oplus t, \\ m_2 = (m_1 \lll b_1) \oplus m_0, \qquad m_3 = (m_2 \lll b_2) \oplus m_1, \\ \mathrm{MW}_{b_1,b_2}(y_1, y_2) = m_2 \oplus m_3. \end{gathered} \tag{3}$$

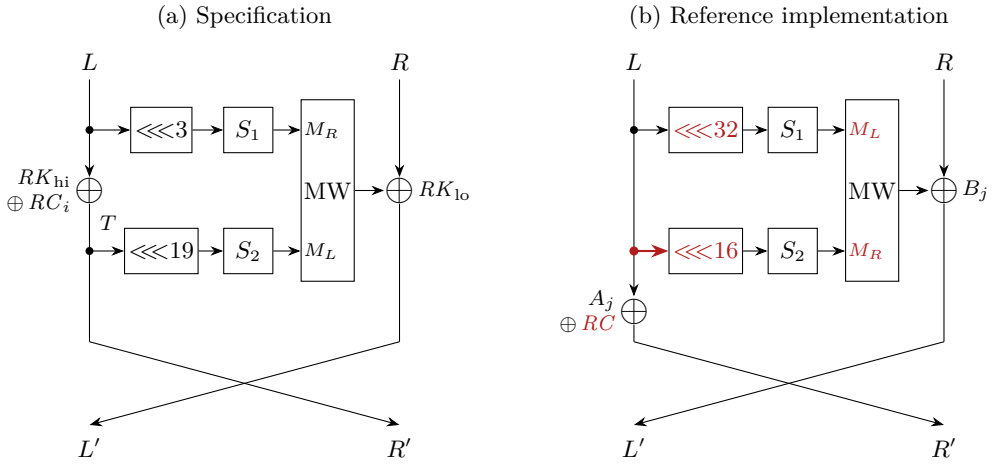

Figure 1: The round function. (a) The specification of the proposal (Figure 2 and Algorithm 1 of [LWP+25]). The input of S_2 is T, which includes the round key and the round constant, and T becomes the next R. RK_{hi} and RK_{lo} are bits 64–127 and bits 0–63 of the round key RK_i. (b) The reference implementation. The rotation amounts are $(32, 16)$, and both S_1 and S_2 read L without a key. The round key A_j and the fixed constant RC are XORed only on the path to the next R. Red marks the differences from the specification. MW is MixWord (Figure 2), and M_L and M_R are the inputs of Eq. (1) of [LWP+25]. In (a), M_L and M_R follow the layout of Figures 2 and 3 of [LWP+25]. The order of the arguments in Algorithm 1 of [LWP+25] gives the same assignment as (b). The proposal does not fix the assignment (Table 2).

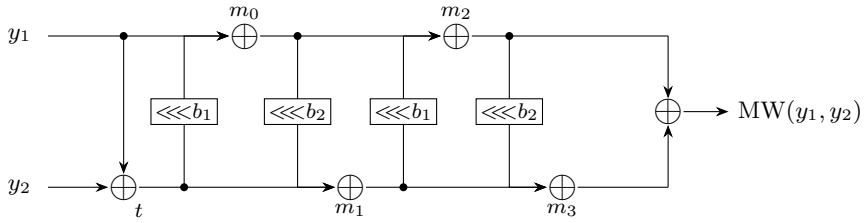

Figure 2: MixWord (Eq. (3)). The upper line starts from y_1, and the lower line starts from $t = y_1 \oplus y_2$. XORs with rotations go back and forth between the two lines four times. The output is the XOR of the two lines. The only rotation amounts are b_1 and b_2. If both are multiples of 16, every bit keeps its position modulo 16.

The reference implementation uses $(b_1, b_2) = (32, 16)$. The ciphertext is (L, R) after 20 rounds. No round key or constant is XORed to the inputs of the S-boxes.

Equation (3) equals Eq. (1) of the proposal with $M_L = y_1$ and $M_R = y_2$. Algorithm 1 of the proposal writes the arguments of MixWord in the order (S_1 output, S_2 output). The reference implementation agrees with this reading. In Figure 2 of the proposal, however, the S_1 output enters the upper input of the MixWord box, and the S_2 output enters the lower input. Figure 3 of the proposal draws M_R as the upper input and M_L as the lower input. If we combine the two figures, M_L can be read as the S_2 output. Thus the proposal does not fix which output is M_L.

Algorithm 1 Comparison of the encryption. (a) Algorithm 1 of the proposal [LWP+25] in its own notation. We merge its last two lines and omit its output line and the index range in line 2. (b) The reference implementation (`encrypt`), written in the same notation. Red marks the differences from (a).

(a) The proposal

Input: $P_{(128)}$, $K_{(128)}$, $T_{(64)}$
1: $P_{(128)} \to L_{0(64)} \,\|\, R_{0(64)}$
2: $GenerateKey(K_{(128)}) \to RK_j$
3: **for** $i = 0$ to $N-1$ **do**
4: $\quad L_i \oplus RK_{i[64\sim127]} \oplus RC_i \to T_i$
5: $\quad SubCell_1(L_i \lll b_1) \to L'_i$
6: $\quad SubCell_2(T_i \lll b_2) \to L''_i$
7: $\quad MixWord(L'_i, L''_i) \oplus RK_{i[0\sim63]} \to L_{i+1}$
8: $\quad T_i \to R_{i+1}$
9: **end for**
10: $L_N \,\|\, R_N \to C_{(128)}$

(b) The reference implementation

Input: $P_{(128)}$, $K_{(128)}$
1: $P_{(128)} \to L_{0(64)} \,\|\, R_{0(64)}$
2: $key_schedule(K_{(128)}) \to (A_i, B_i)$
3: **for** $i = 0$ to 19 **do**
4: $\quad L_i \oplus A_i \oplus RC \to T_i$
5: $\quad SubCell_1(L_i \lll 32) \to L'_i$
6: $\quad SubCell_2(L_i \lll 16) \to L''_i$
7: $\quad MixWord(L'_i, L''_i) \oplus B_i \oplus R_i \to L_{i+1}$
8: $\quad T_i \to R_{i+1}$
9: **end for**
10: $L_{20} \,\|\, R_{20} \to C_{(128)}$

Note on (a): Line 7 does not show the XOR of R_i, but R_i is added in Figure 2 of [LWP+25], as in (b). The input $T_{(64)}$ is not used in the proposal. It is not the value T_i of line 4. Note on (b): $RC =$ `0x243F6A88` $\|$ `0x00000000` is the same in all rounds. Algorithm 2 gives the key schedule.

Algorithm 2 Key schedule of the reference implementation (`key_schedule`)

Input: Master key $K = K_0 \| K_1$
1: $(A, B) \leftarrow (K_0, K_1)$
2: **for** $i = 0$ to 19 **do**
3: $\quad B \leftarrow G^2(B)$
4: $\quad A \leftarrow A \oplus B \oplus (0 \,\|$ `0x243F6A88`$)$
5: $\quad B \leftarrow B \oplus ($`0x85A308D3`$\,\| 0)$
6: $\quad (A_i, B_i) \leftarrow (A, B)$
7: **end for**

The key schedule generates the round keys from the 128-bit master key $K = K_0 \| K_1$ (64 bits each) by the following recurrence. For four 16-bit words (a, b, c, d), with a the most significant, let $G(a, b, c, d) = (a \lll 3 \oplus c,\ b \lll 1 \oplus d,\ a \lll 3,\ b \lll 1)$. Then

$$\begin{aligned} B_0 &= G^2(K_1) \oplus (\texttt{0x85A308D3} \,\|\, 0), \\ A_0 &= K_0 \oplus G^2(K_1) \oplus (0 \,\|\, \texttt{0x243F6A88}), \\ B_{j+1} &= G^2(B_j) \oplus (\texttt{0x85A308D3} \,\|\, 0), \\ A_{j+1} &= A_j \oplus G^2(B_j) \oplus (0 \,\|\, \texttt{0x243F6A88}). \end{aligned} \tag{4}$$

Hence $B_0, \ldots, B_{19}$ depend only on the 64 bits of K_1, and K_0 enters only A_j. The key schedule is an affine map over $\mathbb{F}_2$. We show Algorithm 1 of the proposal and the encryption of the reference implementation side by side, in the same notation, in Algorithm 1. Algorithm 2 shows the key schedule of the reference implementation. The two encryption algorithms differ in five lines (lines 2, 4, 5, 6, and 7).

With $P = 0$ and $K =$ `FF`$\cdots$`FF`, the reference implementation outputs `0C40B9A6 69053EC4 F1DD3EEA 85366E1E`. All five test vectors in Table XVI of the proposal agree with the outputs of the reference implementation. In contrast, none of them agrees with the cipher obtained by changing only the rotation amounts of the reference implementation back to $(3, 19)$ (`tableXVI_check_log.txt`).

Table 2: Differences between the specification of LLBC-128-128 in the proposal [LWP+25] and the reference implementation

Item	Specification of the proposal	Reference implementation
Rotation amounts (b_1, b_2)	$(3, 19)$	$(32, 16)$
Input of S_2	$T_i = L_i \oplus RK \oplus RC_i$ (Algorithm 1 of the proposal)	L_i, with no key and no constant
Round constant	$RC_i = RC_0 \lll i$	`0x243F6A88` in all rounds (XORed to the upper word)
M_L of MixWord	S_1 output by Algorithm 1 of the proposal, S_2 output by the layout of Figures 2 and 3 (not fixed in the proposal)	S_1 output
Key schedule	Eqs. (2)–(4) of the proposal, $RK_0 = K$	Eq. (4). B_j depends only on K_1. The first round key is not the master key.
Constant in the key schedule	—	`0x85A308D3` (RC_0 of LLBC-128-256 in Table III of the proposal)
Test vectors (Table XVI of the proposal)	None of the five agrees with the cipher in which only the rotation amounts are changed back to $(3, 19)$	All five agree
LLBC-128-256	Defined	Not implemented

2.4 Differences from the Specification

Table 2 lists the differences between the specification of the proposal and the reference implementation. In the following sections, the split of the cipher (Section 3) and the full-round distinguisher and decryption (Section 4) depend only on the rotation amounts $(32, 16)$. The complexity of the key recovery (Section 4) depends on the key schedule (4) of the reference implementation.

3 Splitting the Cipher by Its Rotation Amounts

In this section, we give a condition under which an LLBC-type cipher splits into independent ciphers. The number of these ciphers is determined by the greatest common divisor of the rotation amounts. The condition depends only on the rotation amounts. It does not depend on the choice of the S-boxes, the coefficients of the linear layer, the round keys, or the round constants.

3.1 Parts

Definition 1 (Part)**.** For rotation amounts (b_1, b_2), let $g = \gcd(64, b_1, b_2)$, and assume $4 \mid g$. Let $m = g/4$. For $k = 0, \ldots, m-1$, we call the set of bit positions

$$P_k = \{\, i \in \{0, \ldots, 63\} : \lfloor (i \bmod g)/4 \rfloor = k \,\}$$

part k. Each P_k has $64/m$ positions, and $P_0, \ldots, P_{m-1}$ form a partition of $\{0, \ldots, 63\}$. For a 64-bit word x, we write $x[P_k]$ for the bits of x in part k.

For the rotation amounts $(32, 16)$ of the reference implementation, we have $g = 16$ and $m = 4$. Part k consists of the bits i with $\lfloor (i \bmod 16)/4 \rfloor = k$. These are the cells k, $k+4$, $k+8$, and $k+12$ (Figure 3). In the 32-bit words of the C implementation, parts 0, 1, 2, and 3 are the bits of the masks `0x000F000F`, `0x00F000F0`, `0x0F000F00`, and `0xF000F000`. Intuitively, if all rotation amounts are multiples of 16, every bit keeps its position modulo 16 under rotation. The S-box also works within a 4-bit cell, which is a group of such

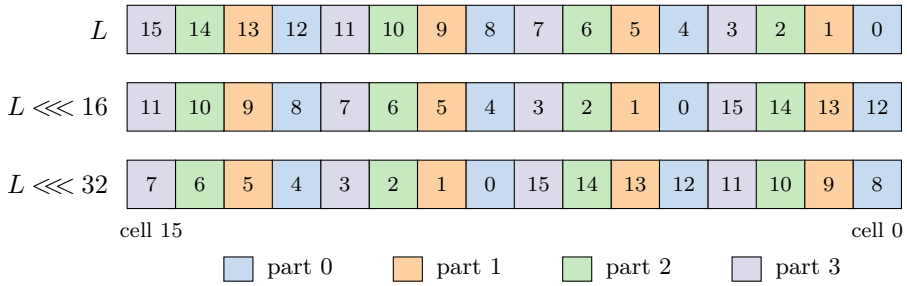

Figure 3: Parts for the rotation amounts $(32, 16)$. Each number is the cell index in the original L, and each color is a part. $L \lll 16$ shifts the cells by four, and $L \lll 32$ shifts them by eight. Hence the color (part) at each position does not change. An S-box works within one cell, so it never crosses the boundary of a part.

positions. Thus the four groups of bits with positions 0–3, 4–7, 8–11, and 12–15 modulo 16 never mix with each other, for any number of rounds.

3.2 The Split Theorem

We state the theorem in the following general form, which covers both the reference implementation and the specification of the proposal. We assume that round j can be written as

$$L' = \lambda\big(S(L \lll b_1 \oplus \gamma_j),\ S'(L \lll b_2 \oplus \delta_j)\big) \oplus R \oplus \beta_j, \qquad R' = L \oplus \alpha_j, \tag{5}$$

where $\alpha_j, \beta_j, \gamma_j, \delta_j$ are arbitrary 64-bit constants (the round keys and round constants together), and S and S' apply arbitrary 4-bit maps cell by cell. Here λ is any linear map that is a composition of rotations by b_1 and b_2 and XORs. The reference implementation is the case $\gamma_j = \delta_j = 0$ and $\lambda = \mathrm{MW}_{32,16}$. The specification of the proposal is the case in which δ_j contains the round key and the round constant.

Theorem 1 (Split)**.** *If $g = \gcd(64, b_1, b_2)$ is a multiple of 4, then any cipher E composed of any number of rounds of the form (5) is a product of $m = g/4$ ciphers:*

$$E(L, R)[P_k] = E_k\big(L[P_k],\ R[P_k]\big) \qquad (k = 0, \ldots, m-1).$$

Here E_k is a permutation on $2 \cdot 64/m$ bits, and it depends only on the bits of part k of the round keys and round constants.

Proof. We show that, in one round, the bits of part k of the output (L', R') depend only on the bits of part k of the input (L, R) and on the constants. The theorem then follows by induction on the number of rounds.

1. **Rotation.** Let r be a multiple of g. Bit i of $x \lll r$ is bit $(i - r) \bmod 64$ of x. Since $g \mid 64$ and $g \mid r$, we have $(i - r) \bmod g = i \bmod g$, and the part does not change. Both b_1 and b_2 are multiples of g.
2. **S-box.** Cell c consists of bits $4c, \ldots, 4c + 3$. Since $4 \mid g$, the value $4c \bmod g$ is a multiple of 4 that is at most $g - 4$. Hence $(4c + t) \bmod g = (4c \bmod g) + t$ for $t = 0, \ldots, 3$, and $\lfloor((4c + t) \bmod g)/4\rfloor = (4c \bmod g)/4$ does not depend on t. Thus each cell lies in one part, and the bits of part k of the S-box output depend only on the bits of part k of its input.
3. **Linear layer.** The map λ is a composition of rotations by b_1 and b_2 and XORs. By item 1, and since XOR does not move bit positions, part k of its output depends only on part k of its input.

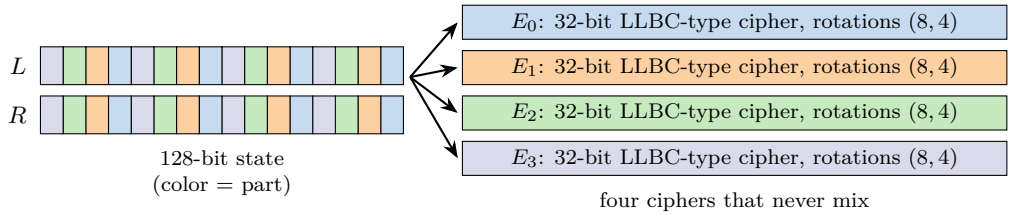

Figure 4: The split of the reference implementation (Corollary 1). The cells k, $k+4$, $k+8$, and $k+12$ of L and R (same color) form part k. The 32-bit cipher E_k maps part k to part k. The four ciphers E_k are LLBC-type ciphers with the same S-boxes and MixWord with rotation amounts $(8, 4)$. They differ only in the round keys and constants.

4. **Constants and the Feistel structure.** The XOR of constants, the XOR with R, and the move of L to R' do not move bit positions.

Hence $(L', R')[P_k]$ depends only on $(L, R)[P_k]$ and on part k of the constants. Since E is a permutation, its restriction to each part is also a permutation. □

Example 1. We follow bit $i = 5$ for $(32, 16)$. Since $5 \bmod 16 = 5$, the bit belongs to part $\lfloor 5/4 \rfloor = 1$. Rotation by 16 moves it to bit 21, and rotation by 32 moves it to bit 37. Both positions are 5 modulo 16, so the bit stays in part 1. Bit 5 belongs to cell 1 (bits 4–7), and all four bits of cell 1 are in part 1. In contrast, for the specification $(3, 19)$, rotation by 3 moves bit 5 to bit 8, which is in part 2. The bit crosses the boundary of a part.

Theorem 1 gives a sufficient condition. For $g = 4$ it gives a single part, so a split into two or more parts needs $8 \mid g$. For the S-boxes and the MixWord of LLBC, an exhaustive check gives the exact number of parts. The number is $g/4$ if $4 \mid g$, and 1 otherwise. Thus, for this cipher, a split occurs if and only if $8 \mid g$.

Proposition 1 (Exhaustive check)**.** *For each of the 3969 pairs with $1 \le b_1, b_2 \le 63$, we build a graph on the bit positions from the dependencies of one round of Eqs. (1)–(3). An edge joins two positions when the new value of one of them depends on the other. The number of connected components of the graph is $g/4$ if $4 \mid g$, and 1 otherwise. A split occurs for 49 pairs. Of these, 40 pairs split into two parts ($g = 8$), 8 pairs split into four parts ($g = 16$), and 1 pair splits into eight parts ($(32, 32)$).*

Proposition 1 is a computational check over all pairs of rotation amounts of a 64-bit word. It found no mismatch with the prediction (`split_condition.py`). We computed the exact dependencies of S_1, S_2, and MixWord, including the terms that cancel. For example, output bit 3 of S_1 does not depend on input bit 0. The simpler graph in which every output bit of a cell depends on all four input bits gives the same numbers.

Corollary 1 (Reference implementation)**.** *The reference implementation of LLBC is a product of four 32-bit ciphers $E_0, \ldots, E_3$. Each E_k is a Feistel cipher with 16-bit halves. It is an LLBC-type cipher with the same S-boxes S_1 and S_2 and with MixWord of rotation amounts $(8, 4)$.*

Proof. We apply Theorem 1 with $g = 16$. We arrange the cells $k + 4j$ $(j = 0, \ldots, 3)$ of part k as nibble j of a 16-bit word. Then rotation by 16 shifts the cells by four, which shifts the nibbles by one (4 bits). Rotation by 32 shifts the nibbles by two (8 bits). □

We checked Corollary 1 with 1000 random pairs of plaintext and key. We compared the encryption function of the reference implementation with the 32-bit part ciphers, which use part k of the round keys and constants. All pairs agreed (`mini_llbc.py`). In contrast, the specification $(3, 19)$ and the rotation amounts $(5, 19)$ of LLBC-128-256 both give $g = 1$, so no split occurs.

Remark 1. Theorem 1 does not assume any key schedule. We gave the round function of the specification the rotation amounts $(32, 16)$ and random round keys. In this test, the input of S_2 includes the round key and the rotating round constant. Following our reading of the figures of the proposal, we take M_L to be the S_2 output. Even then, no influence passed between the parts (`split_spec_variant.py`). Therefore, the weakness of the reference implementation comes only from the choice of the rotation amounts.

Remark 2 (Relation to previous work). Kölbl et al. pointed out that the SIMON-like round function $x \odot S^d(x)$ partitions its output bits into independent classes when $\gcd(d, n) \neq 1$ [KLT15, Section 3.2]. Theorem 1 differs in two points. It treats the whole cipher, not the round function. Its condition also includes the cell boundaries of the S-boxes. For example, $(b_1, b_2) = (2, 6)$ gives $g = 2 > 1$. However, the 4-bit cells connect the even and odd positions, so no split occurs (Proposition 1).

4 Attacks on All Rounds

In this section, we use the split of Corollary 1 to build three attacks on all 20 rounds of the reference implementation (Table 1). The distinguisher and the decryption depend only on the rotation amounts. They hold for any number of rounds, any S-boxes, and any key schedule. The key recovery uses a property of the key schedule (4) of the reference implementation. We count time complexity in 20-round encryptions. We ran the experiments in WSL on an Intel Core i5-13500 (20 threads). As the encryption oracle, we used the `encrypt` function of the reference implementation without any change.

4.1 A Distinguisher with Probability 1

We choose a plaintext P and a plaintext P' that differs from P only in the bits of part 0. We obtain their ciphertexts C and C'. By Theorem 1, parts 1–3 of C and C' (96 bits) always agree. For a random permutation, this agreement happens only with probability about 2^{-96}. Thus two chosen plaintexts distinguish the full 20-round reference implementation from a random permutation with probability 1.

4.2 Decryption of Any Ciphertext with 2^{32} Chosen Plaintexts

Each part cipher E_k is a 32-bit permutation. We obtain its full table (codebook) from 2^{32} inputs. The same 2^{32} chosen plaintexts give the full tables of all four parts at once. For $v \in \{0, 1\}^{32}$, let $P(v)$ be the plaintext whose four parts all equal v, that is, $P(v)[P_k] = v$ for $k = 0, \ldots, 3$. Part k of the ciphertext of $P(v)$ is $E_k(v)$. Querying all v gives the full tables of $E_0, \ldots, E_3$ (Figure 5). We then decrypt any ciphertext C part by part. For each part k, we look up the value v with $E_k(v) = C[P_k]$ in the table. The same tables also encrypt any plaintext. Thus, without the key, the attacker in effect obtains the full codebook of the whole cipher (2^{128} entries).

Storing all tables needs 4 parts $\times\, 2^{32}$ entries $\times\, 32$ bits $= 2^{39}$ bits (64 GiB). If the target ciphertexts are given in advance, the attacker only needs to compare these values during the queries. The memory is then proportional to the number of target ciphertexts.

Experiment A. Under a random secret key, we took the ciphertexts of 1000 random plaintexts as the targets. The targets were given before the queries, so we compared them during the queries and did not store the 64-GiB tables. We recovered the plaintexts from the ciphertexts only, and we used the plaintexts only to check the answers. The 2^{32} queries took 175.5 seconds, and the peak memory was 4.8 MB. All 1000 plaintexts were recovered correctly. In each part, each of the 1000 target values appeared exactly once. This agrees with the fact that each E_k is a permutation.

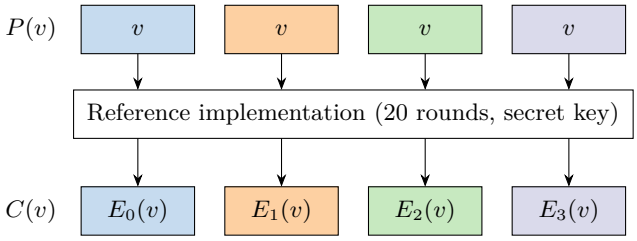

Figure 5: Decryption with 2^{32} chosen plaintexts. We query the plaintexts $P(v)$, whose four parts all equal v, for $v = 0, 1, \ldots, 2^{32} - 1$. This builds the full tables (2^{32} rows each) of the four part ciphers E_k at once. We decrypt any ciphertext C by looking up, for each part k, the value v with $E_k(v) = C[P_k]$.

Table 3: Number of dimensions of the master key on which part 0 depends (the rank of $\pi_0 \circ \mathcal{L}$). The other parts have the same values.

Rounds	1	2	3	4	5	6	8	10	12	13–20
Rank	32	40	48	56	64	66	70	74	78	80

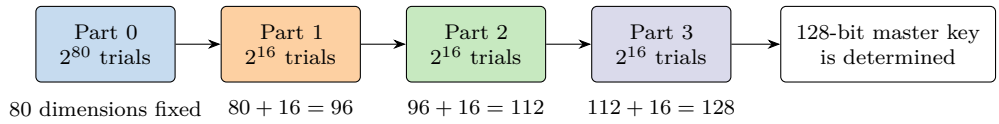

Figure 6: Flow of the master-key recovery. Each stage checks the corresponding part of the three known pairs. Part 0 fixes 80 dimensions of the master key. Each of parts 1–3 then needs to try only 16 new dimensions.

4.3 Master-Key Recovery

The key schedule (4) of the reference implementation is an affine map over $\mathbb{F}_2$. Thus the concatenation of the round keys of r rounds can be written as $\mathrm{rk}(K) = \mathcal{L}(K) \oplus \mathrm{rk}(0)$. Here $\mathcal{L}$ is a linear map from the 128 bits of the master key to the $128r$ bits of the round keys. Part k uses only the bits of part k of the round keys. Let π_k denote this projection. Then part k depends on the master key only through $\pi_k \circ \mathcal{L}$. Table 3 shows the rank of $\pi_0 \circ \mathcal{L}$ for each number of rounds. For all 20 rounds, the rank for each part is 80, which consists of all 64 dimensions of K_1 and 16 dimensions of K_0. The rank is 96 for two parts, 112 for three parts, and 128 for four parts.

The attack uses three known pairs $(P^{(i)}, C^{(i)})$ and narrows down the key part by part.

1. **Part 0.** For each of the 2^{80} values in the image of $\pi_0 \circ \mathcal{L}$, we encrypt part 0 of the three plaintexts with E_0. We compare the results with part 0 of the ciphertexts. This comparison is a $3 \times 32 = 96$-bit condition. We assume that the parts of the three plaintexts are distinct and that a wrong value gives outputs that behave like random values. This is the usual wrong-key assumption. Then the expected number of wrong candidates that remain is $2^{80-96} = 2^{-16}$. The remaining value restricts the master key to a coset of $\ker(\pi_0 \circ \mathcal{L})$, which has 48 dimensions.
2. **Parts 1–3.** Within this coset, we try the 2^{16} values of the 16 new dimensions on which part 1 depends. We check them with part 1 of the same three pairs. We do the same for parts 2 and 3, with 16 dimensions each. Then the master key is determined (Figure 6). If more than one candidate remains at a stage, we keep all of them for the next stage.

One trial is one evaluation of a 32-bit cipher E_k, which costs about 1/4 of a 20-round encryption. Most candidates are rejected by the first pair. Even a candidate that is tested on all three pairs costs 3/4 of an encryption, which is less than one encryption. Thus, under the assumption above, the time complexity is at most $2^{80} + 3 \cdot 2^{16}$ encryptions, and this bound is conservative. It is below exhaustive search (2^{128}) by a factor of 2^{48}. The ranks in Table 3 are exact, and the number of surviving candidates is an estimate.

Experiment B. We cannot run 2^{80} computations, so we ran a reduced version. We gave the attacker 48 of the 80 dimensions of part 0 and left $d = 32$ dimensions unknown. We did not reduce the stages for parts 1–3, and we ran them at the real scale (2^{16} values each). The stage for part 0 took 150.8 seconds for 2^{32} trials, and only the true value remained. For parts 1–3, one candidate remained at each stage. The recovered 128-bit master key was equal to the secret key, and it also encrypted a fourth known plaintext correctly. The peak memory was 4.4 MB. The ranks 80, 96, 112, and 128 were consistent with the kernel dimensions 48, 32, 16, and 0 computed in the experiment. In each case, the rank and the kernel dimension sum to 128.

Remark 3. The complexity of the key recovery depends on the key schedule. If we combine the round function of the reference implementation with the key schedule of the specification (Eqs. (2)–(4) of the proposal), each part depends on 118 dimensions of the master key (`spec_key_rank.py`). In contrast, the distinguisher and the decryption do not depend on the key schedule.

5 Standard Techniques on the Reference Implementation

In this section, we study what standard techniques show for the reference implementation. We consider differential [BS91], linear [Mat94], integral [DKR97, KW02], impossible differential [BBS99], and zero-correlation [BW12] cryptanalysis. First, we show that the assumption of the usual models, which treat the S-boxes as independent, does not hold for the reference implementation, and we give the correct model (Section 5.1). Next, we show that distinguishers based on properties with probability 1 hold for all 20 rounds. These are impossible differential, zero-correlation, integral, and truncated differential distinguishers. A standard automatic search also returns them as 20-round distinguishers (Section 5.2). In contrast, classical differential and linear attacks chain characteristics of small probability. The characteristics and the integral properties that we examined distinguish only 8 to 10 rounds inside one part (Section 5.3).

5.1 Joint S-Boxes: S_1 and S_2 Read the Same Nibble

In the reference implementation, the outputs of S_1 and S_2 in the same round are functions of the same input. Cell c of S_1 reads four bits of $L \lll 32$, and cell c of S_2 reads four bits of $L \lll 16$. Both 32 and 16 are multiples of 4. Thus each nibble of L enters exactly one cell of S_1 and one cell of S_2, and no key or constant is XORed in between. Therefore the outputs of S_1 and S_2 are not independent. In terms of differential cryptanalysis, the same nibble difference enters S_1 and S_2 at the same time. The output differences of the two S-boxes cannot be chosen independently.

The usual differential and linear models treat the S-boxes as independent of each other. They take the product of the probabilities (correlations) of the S-boxes as the probability (correlation) of a characteristic. This treatment is correct when the S-boxes read different bits, or when uniform round keys are XORed between the S-boxes. If the round keys are independent and uniform, the probability of a characteristic averaged over the keys is the product of the probabilities of the rounds [BS91, Theorem 1], [LMM91, Theorem 2]. In the SIMON round function, one nonlinear part reads two rotations of the same word. For

Table 4: Probabilities of pairs of output differences for a nibble with input difference 4, counted over the 16 values of x

Output differences of (S_1, S_2)	Independent model	Actual
$(4, 1)$	1/16	0
$(8, 4)$	1/16	0
$(3, 4)$	1/32	1/8
$(4, 6)$	1/32	1/8
Number of possible pairs	36	8

Table 5: Check of the S-box models. Measurements over all 2^{32} inputs of one part with four keys. For differentials, the values are the numbers of inputs x that satisfy the input and output differences of the characteristic. For linear approximations, the values are correlations.

Model	Type	Rounds	Weight	Predicted	Measured
Independent	Differential	6	24	256	0–2
Joint	Differential	3	6	$2^{26} = 67{,}108{,}864$	67,108,864 (all keys)
	Differential	5	12	$2^{20} = 1{,}048{,}576$	1,045,790–1,055,730
	Linear	5	4	$\pm 2^{-4} = \pm 0.0625$	± 0.06246 to ± 0.06257
	Linear	6	7	$\pm 2^{-7} \approx \pm 0.007813$	± 0.007825 to ± 0.007883

this case, the exact one-round probability that takes the dependence into account was derived separately [KLT15, Sections 3–4]. In the specification of the proposal, a round key is XORed to the input of S_2. This is the second case above. For the differential probability of one round, the average over a uniform key nibble equals the product of the probabilities of the two S-boxes exactly. This identity concerns differentials and one round. We make no claim here about a fixed key, about several rounds under the actual key schedule, or about linear correlations. The reference implementation removes this round key. Therefore the assumption of the independent model fails.

The correct way is to treat the nibble map $x \mapsto (S_1(x), S_2(x))$ as one function from 4 bits to 8 bits. We call this the *joint model*. In the difference distribution table of this function, each nonzero input difference has exactly 8 possible pairs of output differences. Each of them has probability 2/16 (weight 3). Here the weight of a differential characteristic with probability p is $-\log_2 p$. In the linear approximation table, there are 7 nonzero pairs of masks with correlation ± 1. Table 4 compares the two models for a nibble with input difference 4. The pairs with the highest probability in the independent model never occur. The pairs that do occur have a higher probability than in the independent model.

We confirmed this difference by measurement. We used all 2^{32} inputs of a 32-bit part (Corollary 1) and the round keys of part 0 generated by the key schedule of the reference implementation. For differentials, we counted the inputs that satisfy the input and output differences of a characteristic. For linear approximations, we computed the correlation (Table 5). These measured values include every characteristic with the same input and output. The best characteristic of the independent model holds only about 1/100 as often as predicted, or less. The characteristics of the joint model agree well with the predictions.

5.2 Distinguishers with Probability 1 for All Rounds

Theorem 1 gives the following four distinguishers for any number of rounds. We state each of them for two parts $j \neq k$.

Table 6: Automatic search on 20 rounds of the whole 128-bit cipher (STP, joint model). We fixed the input and the output and checked whether a characteristic connects them.

Input → output (one bit each)	Differential (impossible differential search)	Linear (zero-correlation search)
Part 0 → part 1	no solution (2011 s)	no solution (563 s)
Part 0 → part 2	no solution (2133 s)	no solution (561 s)
Part 0 → part 3	no solution (1300 s)	no solution (546 s)
Part 3 → part 0	no solution (1697 s)	no solution (534 s)
Control: best 32-bit characteristics (3 and 5 rounds)	solution found	solution found

- **Truncated differential:** If the input difference is limited to part j, the output difference is limited to part j with probability 1.
- **Impossible differential:** A differential with an input difference only in part j and an output difference in part k never occurs.
- **Zero-correlation linear approximation:** A linear approximation with the input mask in part j and the output mask in part k has correlation exactly 0.
- **Integral:** For a set of plaintexts that vary only the bits of part j, part k of the ciphertexts is constant.

These are restatements of one fact: no influence passes between the parts. They are essentially the same as the distinguisher of Section 4.1.

A standard automatic search also finds these distinguishers for all 20 rounds. We modeled 20 rounds of the whole 128-bit reference implementation with STP [GD07]. We fixed the input and the output (differences or masks), and we checked whether a characteristic that connects them exists. This follows the same idea as the MILP-based search for impossible differentials by Sasaki and Todo [ST17]. An impossible differential found with the concrete S-box tables is impossible for any key schedule [ST17, Observation 1]. We modeled the S-boxes with the joint model of Section 5.1. We used one bit for the input and one bit for the output, and we checked four pairs that cross parts (Table 6). For all four pairs, both the differential model and the linear model had no solution. Thus the automatic search gave 20-round impossible differentials and 20-round zero-correlation approximations. As a control, we placed the best 32-bit characteristics of Section 5.3 in part 0. For both 3 and 5 rounds, the solver found a solution. This confirms that the model does not always return "no solution".

5.3 Rounds Reached by Classical Differential, Linear, and Integral Attacks

Under the split, a 128-bit characteristic splits into characteristics of the parts, and its weight is the sum of the weights of the parts. The four parts do not affect each other, and they are the same cipher except for the keys and constants. Thus the lightest characteristic uses only one part. The following holds.

Proposition 2. *The minimum weight of differential (linear) characteristics of the reference implementation equals the minimum weight of differential (linear) characteristics of the 32-bit cipher E_k.*

Proof. A nonzero characteristic is nonzero in at least one part. If we keep only that part and set the other parts to zero, we obtain a valid characteristic whose weight is not larger.

Table 7: Minimum weights of the 32-bit version in the joint model. For 12 rounds (linear), we only confirmed that no characteristic of weight 15 or less exists. For 20 rounds, the values are lower bounds given by sums of the values for fewer rounds ($8+6+6$ rounds for differentials, $12+8$ rounds for linear characteristics). We did not compute the differential weights for 11 and 12 rounds.

Rounds	2	3	4	5	6	7	8	9	10	11	12	20
Differential	3	6	9	12	21	24	27	30	33			≥ 69
Linear	1	2	3	4	7	8	10	12	12	14	≥ 16	≥ 26

The characteristic of each part is a characteristic of E_k. Keys and constants do not affect how differences propagate or the size of correlations. □

Based on this proposition, we computed the minimum weights on the 32-bit version. In the independent model, the minimum differential weights of the 32-bit version (2 to 10 rounds) agreed with those of the 128-bit version for all numbers of rounds. This is consistent with the proposition. Table 7 shows the minimum weights in the joint model. For each value w without $\geq$, we confirmed with STP that a characteristic of weight w exists and that no characteristic of weight $w-1$ or less exists. We followed [MP13, KLT15] for the search of characteristics with STP. The weight of a linear characteristic with correlation c is $-\log_2 |c|$.

The minimum numbers of differentially active S-boxes follow from Table 7. In the joint model, each active nibble of L activates one cell of S_1 and one cell of S_2, and it has weight exactly 3. Hence a differential characteristic of weight w has $2w/3$ active S-boxes, counted in both S-box layers. Over the characteristics with nonzero probability, the minimum numbers of active S-boxes for 2 to 7 rounds are 2, 4, 6, 8, 14, and 16. This count needs the joint model. A model that treats S_1 and S_2 as independent also admits characteristics that have probability 0 in the reference implementation (Section 5.1).

Inside one part, a single differential characteristic is usable only if its weight is below about 32, and a single linear characteristic only if its weight is below about 16. The two limits differ because the linear weight is the weight of the correlation. A differential characteristic with probability 2^{-w} needs about 2^w pairs. A linear characteristic with correlation $\pm 2^{-w}$ needs about 2^{2w} known plaintexts. Hence, for a 128-bit block cipher, a differential characteristic is usually regarded as usable if $w < 128$, and a linear characteristic if $w < 64$. In the reference implementation, however, a characteristic splits into characteristics of the parts, and each part has only 2^{32} inputs. The limits therefore become 32 and 16. As a small example, a 4-bit part has only 16 inputs, and one input difference gives only 8 pairs. Thus almost surely no pair satisfies a characteristic with probability 2^{-10}. These limits are the usual criteria for a single characteristic. They do not bound attacks that use many characteristics with the same input and output, that is, differentials and linear hulls.

More precisely, for differentials, the expected number of inputs x that satisfy given input and output differences is 2^{32-w} for a characteristic with probability 2^{-w}. This expectation is an average over independent and uniform round keys. For a random permutation, it is 1. For linear approximations, the correlation of a random 32-bit permutation fluctuates with size about 2^{-16}. A correlation must be clearly larger than this to be usable. We therefore measured the input and output of the best characteristics over all 2^{32} inputs of a part (Table 8). In the table, the predicted value is that of the single characteristic. The measured value counts every characteristic with the same input and output differences under each key. In the same way, a measured correlation is that of the linear hull.

For differentials, the measured counts deviate from the predictions under the actual key schedule. For 7 rounds, the count exceeds the prediction. Other characteristics with the same input and output differences and the dependence on the key can both contribute. For 8 rounds, the count is below the prediction. For 9 rounds, 10 to 16 inputs satisfy the input and output differences. This is clearly more than 1 for a random permutation.

We also fixed one input difference and counted the whole distribution of output differences. For a random permutation, the number of pairs with a given output difference follows a Poisson distribution with mean 1/2. Over the 2^{32} output differences, the largest number is then 9 or 10 pairs, that is, 18 or 20 inputs, with probability about 97%. It is 11 pairs, that is, 22 inputs, with probability about 3%. Twenty rounds of the part cipher gave 18 or 20 inputs for four input differences. For 9 rounds, the most frequent output difference was produced by 40 inputs for one of six input differences. For 10 rounds, the maximum was 18 to 22 inputs for all seven input differences that we tried, and we could not distinguish them from a random permutation.

For linear approximations, the correlations of 10 rounds, $2^{-11.2}$ to $2^{-11.7}$, are clearly larger than the fluctuation 2^{-16}. For 11 rounds, the correlation varies with the key from $2^{-14.3}$ to $2^{-17.2}$, which is about the same as the fluctuation. For 12 or more rounds, the minimum weight is at least 16. Hence the correlation of any single characteristic does not exceed 2^{-16}. We did not study how much linear hulls increase it.

For integrals, we fixed one input bit and took all values of the remaining 31 bits (data 2^{31}). For each of the 32 choices of the fixed bit, we counted the output bits whose XOR sum is zero for every one of 16 master keys. For 7 rounds, all 32 output bits have a zero sum when the fixed bit is in the left half. The 16 output bits of the right half have a zero sum when the fixed bit is in the right half. For 8 rounds, the 16 output bits of the right half have a zero sum when the fixed bit is in the left half, and no output bit otherwise. For 9 rounds, no output bit has a zero sum for all 16 keys, for any choice of the fixed bit. For a random permutation, an output bit has a zero sum with probability 1/2 for each key, and thus with probability 2^{-16} for all 16 keys.

From these results, inside one part, we distinguished 9 rounds with differentials, 10 rounds with linear approximations (11 rounds is borderline), and 8 rounds with integrals (Table 9). These numbers are observations for the characteristics, the input differences, and the keys that we examined. They are not upper bounds for these techniques.

5.4 A Pitfall of Looking Only at 128 Bits

For a 128-bit block cipher, a full-round differential characteristic of weight less than 128 is usually regarded as a full-round distinguisher. For a linear characteristic, the limit is 64. For the reference implementation, Table 7 gives lower bounds of 69 (differential) and 26 (linear) for 20 rounds. These are far below 128 and 64. Thus 20-round characteristics below these limits may well exist. If we look only at the whole 128 bits, such a characteristic looks like a full-round differential or linear distinguisher. However, such a characteristic splits into characteristics of the parts. A differential characteristic of weight less than 128 stays inside one part, because two parts already need weight at least 138. Even if a differential characteristic of weight 69 existed, the expected number of inputs x that satisfy it would be 2^{32-69} on average over the keys. A linear characteristic has weight at least 26 in every part that it uses, which is above the limit of 16 for a 32-bit part. As single characteristics, they are useless for attacks. We did not compute the minimum weight of 20 rounds itself in the joint model.

In fact, in our early search on the whole 128 bits, we found a 20-round differential characteristic of weight 76 in the independent model. Its differences were only at bit positions $\equiv 14 \pmod{16}$ in all rounds, so it stayed inside part 3. The fact that its differences never left one class of bit positions was the clue that led us to the split.

Table 8: Measurements over all 2^{32} inputs of one part, with the best characteristics of the joint model. Differentials and linear approximations use four keys. The output-difference distribution uses one key. Integrals use data 2^{31} (one bit fixed) and 16 keys. An entry counts the output bits whose sum is zero for all 16 keys, and the table shows the maximum over the 32 choices of the fixed bit. The last column is the value expected for a random 32-bit permutation.

Method	Rounds	Weight	Predicted	Measured	Random
Differential	6	21	2048	1970–2156	1
(number of inputs)	7	24	256	1642–3112	1
	8	27	32	2–6	1
	9	30	4	10–16	1
	10	33	0.5	0–2	1
Output-difference	9	—	—	40[a]	18–20
distribution (largest number of inputs)	10	—	—	18–22[b]	18–20
Linear	7	8	2^{-8}	$2^{-8.4}$ to $2^{-7.7}$	2^{-16}
(|correlation|)	8	10	2^{-10}	$2^{-10.0}$ to $2^{-9.7}$	2^{-16}
	9	12	2^{-12}	$2^{-12.0}$	2^{-16}
	10	12	2^{-12}	$2^{-11.7}$ to $2^{-11.2}$	2^{-16}
	11	14	2^{-14}	$2^{-17.2}$ to $2^{-14.3}$	2^{-16}
Integral	7	—	—	32 / 32	0
(bits with zero sum	8	—	—	16 / 32	0
for all 16 keys)	9	—	—	0 / 32	0

[a] 40 for one of six input differences, and 18 to 20 for the other five.
[b] For all seven input differences that we tried.

Table 9: Number of rounds distinguished by each technique on the reference implementation (20 rounds in total). The last three rows are observations for the examined characteristics and structures, not upper bounds.

Technique	Rounds distinguished	Evidence
Split (Theorem 1)	20 (all)	proof, Experiments A and B
Impossible differential and zero correlation (automatic search)	20 (all)	STP, four pairs each
Classical linear	10 (11 is borderline)	minimum weight, full measurement
Classical differential	9	minimum weight, full measurement, output-difference distribution
Integral (data 2^{31})	8	full measurement

An automatic search shows symptoms, such as full-round distinguishers and light characteristics. It does not show their cause, which is the split. It also does not check automatically whether the assumptions of its model, such as the independence of the S-boxes, hold for the evaluated implementation. The reference implementation is an example in which both points matter.

Table 10: The security expected for a 128-bit block cipher and the actual security of the reference implementation (all 20 rounds)

Item	Expected level	Reference implementation
Best attack evaluated by the designers	9-round integral attack (on the specification)	All 20 rounds (this paper)
Distinguishing from a random permutation	—	2 chosen plaintexts, probability 1
Decryption of any ciphertext (chosen plaintexts)	2^{128}	2^{32} chosen plaintexts (175.5 s on a PC)
Decryption of any ciphertext (known plaintexts)	2^{128}	92.9% with 2^{34} known plaintexts, 99.9% with 2^{35} (estimate)
Master-key recovery	2^{128} (exhaustive search)	3 known plaintexts, $2^{80} + 3 \cdot 2^{16}$ encryptions

6 Discussion

6.1 The Actual Security of the Reference Implementation

Table 10 compares the security expected for a 128-bit block cipher with the actual security of the reference implementation. All 20 rounds of the reference implementation are broken with practical complexity. **In particular, decrypting any ciphertext needs only 2^{32} chosen plaintexts, which is below the level expected for a cipher with a 128-bit key (2^{128}) by a factor of 2^{96}.** Recovering the master key also needs less computation than exhaustive search, by a factor of 2^{48}.

The row for known plaintexts comes from the following estimate. Under one fixed secret key, from N random known plaintexts, on average $2^{32}(1 - e^{-N/2^{32}})$ rows of the full table of each part E_k are filled. A given ciphertext can be decrypted when the values of all four of its parts appear in the tables. This happens with probability $(1 - e^{-N/2^{32}})^4$, which is 0.929 for $N = 2^{34}$ and 0.999 for $N = 2^{35}$. We checked this formula with a small-scale experiment in which each part has 2^{12} values ($N = 4 \cdot 2^{12}$, 2000 ciphertexts). Over ten trials, the mean success rate was 0.929 (0.914 to 0.943), which agrees with the computed value 0.929. Thus, even when the attacker cannot choose plaintexts, 2^{34} plaintext–ciphertext pairs (512 GiB) let the attacker decrypt most later ciphertexts without recovering the key.

6.2 Risks of a Mismatch between the Specification and the Release

The proposal states that the code and the test vectors are released for reproducibility [LWP+25, Appendix]. It is natural for users and evaluators to use this code as a model and to check their own implementations with the test vectors. However, the test vectors of the proposal agree with the reference implementation, not with the specification (Section 2). Thus an implementation that matches the test vectors is not the cipher of the specification. It is the cipher whose 20 rounds we break in this paper.

Conversely, a user who implements the specification notices that the test vectors do not match. However, the proposal alone does not tell whether the cause lies in the specification or in the release. As Table 2 shows, the two differ not only in the rotation amounts. They differ in eight points, including the input of S_2, the round constants, and the key schedule. Moreover, the assignment of the inputs of MixWord (M_L) is not fixed even within the proposal.

Neither the release nor the proposal explains why the reference implementation uses values that differ from the specification. Whatever the reason, the change of the rotation amounts from $(3, 19)$ to $(32, 16)$ splits the cipher into four 32-bit ciphers, and all 20 rounds are broken.

6.3 Relation to the Designers' Security Evaluation

The designers evaluated the specification of the proposal against integral, differential, linear, and other attacks, and they set the number of rounds from this evaluation. Our results say nothing about the security of the specification itself. No split occurs for the rotation amounts $(3, 19)$ of the specification (Proposition 1). We study differential attacks on the specification in a separate paper [KI26]. The issue is that the target of the evaluation differed from the release.

If standard evaluations had been applied to the release, the weakness would have been found easily. As shown in Section 5, a standard automatic search gives 20-round impossible differentials and zero-correlation approximations for the whole 128 bits, and each pair took 9 to 36 minutes. Moreover, if we trace, from the description of the cipher, which plaintext bits each ciphertext bit can depend on, we see in less than one second that the reference implementation never reaches full diffusion. We note that the conclusion of the proposal states that LLBC appears to resist impossible differential and zero-correlation attacks [LWP+25]. That statement concerns the specification. In contrast, for the reference implementation, a standard automatic search gives 20-round distinguishers of both kinds.

However, evaluating the release is not enough by itself. As shown in Section 5.1, the usual differential and linear models that treat the S-boxes as independent rely on an assumption that fails for the reference implementation. A judgment of security from the minimum weights alone can be wrong. Moreover, if we look only at the whole 128 bits, a useless characteristic inside one part looks like a full-round distinguisher (Section 5.4).

6.4 Recommendations for Designers and Evaluators

From our experience, we recommend the following checks. All of them are cheap, and they can be done before a design is published. After each item, we show what the check reveals for the reference implementation of LLBC.

1. **Check the release with test vectors made from the specification.** Generate the test vectors with an implementation made from the specification, independently of the released code. Then confirm that the released code reproduces them. *For LLBC:* If we change only the rotation amounts back to the specification, all five test vectors of the proposal fail (Section 2). This check would have revealed, before publication, that the release does not follow the specification (Table 2).
2. **Check the structure of the release itself.** Trace which plaintext bits each bit of the released code can depend on. Find the number of rounds after which every ciphertext bit depends on every plaintext bit (the number of rounds for full diffusion). If the dependency graph between the plaintext bits and the ciphertext bits has two or more components, the cipher splits. *For LLBC:* The reference implementation never reaches full diffusion, and it splits into four components. This takes less than one second to see.
3. **Check the assumptions of the evaluation models.** When the security is evaluated with automatic search, confirm that the assumptions of the model, such as the independence of the S-boxes, hold for the evaluated implementation. *For LLBC:* The best 6-round differential characteristic of the independent model holds for only 0 to 2 inputs, while 256 are predicted (Table 5).
4. **State the correspondence between the release and the paper.** State which version of the specification in the paper the release (commit or archive hash) corresponds to. If the release differs from the specification, list the differences with their reasons. Also state the conditions for redistribution and modification (license). *For LLBC:* The release states neither its correspondence to the specification nor the conditions for redistribution.

6.5 Significance of This Work

We summarize the significance of this work in four points.

- **A practical full-round break of the official release of a published lightweight cipher.** To the best of our knowledge, this paper gives the first full-round attack on the reference implementation of LLBC. We ran the decryption at full scale on a PC, and it finished in 175.5 seconds.
- **A condition for the split in terms of the rotation amounts only.** The condition $8 \mid \gcd(64, b_1, b_2)$ for a split does not depend on the S-boxes or on the coefficients of the linear layer. Designers and implementers of LLBC-type ciphers can use it directly as a rule for choosing rotation amounts.
- **Two concrete ways in which automatic evaluation can mislead.** First, a useless characteristic inside one part can look like a full-round distinguisher. Second, the assumption that the S-boxes are independent can fail in an implementation. Automatic search is now widely used for security evaluation. We showed with concrete numbers why the target and the model assumptions of an evaluation must be checked.
- **A design lesson.** In the specification of the proposal, a round key is XORed to the input of S_2. Thus, for the differential probability of one round, the two S-boxes behave independently on average over the key. The reference implementation changed the rotation amounts, which split the cipher. It also removed this round key, which broke the assumption of the standard models. A key XOR before an S-box mixes in the key, and it also supports the assumptions of the evaluation.

6.6 Limitations

- Theorem 1 and the structure check above treat only splits along coordinate bits. They do not treat splits that appear after a change of the linear basis. Such structures relate to the invariant subspace attack, which uses a subspace that is invariant under the round function [LAAZ11].
- The key-recovery complexity $2^{80} + 3 \cdot 2^{16}$ comes from a naive method that tries all round-key values part by part. It may be improved.
- The numbers of rounds reached by the classical differential and linear attacks (Section 5.3) are based on the best characteristics and on measurements of their inputs and outputs. The results for the output-difference distribution and for integrals are observations from measurements. We did not test all input differences. For integrals, we tested only the sets that fix one input bit.
- The success rate of the decryption with known plaintexts (Section 6.1) is a computed estimate. We did not run it at full scale.

6.7 Disclosure to the Designers

We informed the corresponding author of the proposal and the author of the reference implementation of our results by e-mail on the day we submitted this paper to the IACR ePrint Archive.

7 Conclusion

We analyzed the reference implementation released by the designers of LLBC. We showed that it differs from the specification of the proposal in eight points. In particular, the reference implementation uses the rotation amounts $(32, 16)$ instead of $(3, 19)$ of the

specification. We proved that, as a result, the 128-bit cipher is a product of four independent 32-bit ciphers. In general, the cipher is a product of $g/4$ ciphers when $4 \mid g$ for $g = \gcd(64, b_1, b_2)$. Thus it splits when $8 \mid g$, and this condition depends only on the rotation amounts.

With this split, we attacked all 20 rounds. We showed a distinguisher with probability 1 from two chosen plaintexts, the decryption of any ciphertext with 2^{32} chosen plaintexts, and master-key recovery from three known plaintexts with $2^{80} + 3 \cdot 2^{16}$ encryptions. We ran the decryption at full scale on a PC, and it recovered the plaintexts of all 1000 ciphertexts in 175.5 seconds.

Even when the attacker cannot choose plaintexts, 2^{34} known plaintexts are enough to decrypt an estimated 92.9% of arbitrary ciphertexts without the key. The reference implementation is not secure as a cipher, and it should not be used. The test vectors of the proposal agree with the reference implementation. Thus users who follow the released code and rely on these test vectors also end up with the same cipher. In particular, we recommend two checks when a design is published. The first check compares the release with test vectors made independently from the specification. The second check confirms the structure of the release itself (the number of rounds for full diffusion). Future work includes the detection of splits that appear after a change of the linear basis, a better complexity for the key recovery, and a systematic study of mismatches between specifications and releases.

References

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In *EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 12–23. Springer, 1999.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[BW12] Andrey Bogdanov and Meiqin Wang. Zero correlation linear cryptanalysis with reduced data complexity. In *FSE 2012*, volume 7549 of *LNCS*, pages 29–48. Springer, 2012.

[DKR97] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher Square. In *FSE 1997*, volume 1267 of *LNCS*, pages 149–165. Springer, 1997.

[FSC+26] Lukas Fluri, Avital Shafran, Nicholas Carlini, Matthew Jagielski, Milad Nasr, Orr Dunkelman, Eyal Ronen, and Florian Tramèr. CryptanalysisBench: Can LLMs do cryptanalysis? arXiv:2607.18538v2, 2026.

[GD07] Vijay Ganesh and David L. Dill. A decision procedure for bit-vectors and arrays. In *CAV 2007*, volume 4590 of *LNCS*, pages 519–531. Springer, 2007.

[Git25] LLBC-cipher. GitHub repository of 13est-Wishes, https://github.com/13est-Wishes/LLBC-cipher, 2025. Commit f30143d (June 22, 2025).

[GLL21] Ying Guo, Lang Li, and Botao Liu. Shadow: A lightweight block cipher for IoT nodes. *IEEE Internet of Things Journal*, 8(16):13014–13023, 2021.

[KI26] Sei Kofuku and Yasutaka Igarashi. Differential cryptanalysis of the lightweight block cipher LLBC-128-128. Submitted to IEEE Internet of Things Journal, 2026.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON block cipher family. In *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 161–185. Springer, 2015.

[KSK+25] Sunyeop Kim, Myoungsu Shin, Seonkyu Kim, Hanbeom Shin, Insung Kim, Donggeun Kwon, Dongjae Lee, Seonggyeom Kim, Deukjo Hong, Jaechul Sung, and Seokhie Hong. Redefining security in Shadow cipher for IoT nodes: New full-round practical distinguisher and the infeasibility of key-recovery attacks. *IEEE Internet of Things Journal*, 12(6):6805–6817, 2025.

[KW02] Lars Knudsen and David Wagner. Integral cryptanalysis. In *FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, 2002.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of PRINTcipher: The invariant subspace attack. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 206–221. Springer, 2011.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *EUROCRYPT 1991*, volume 547 of *LNCS*, pages 17–38. Springer, 1991.

[LNP15] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of KLEIN. In *FSE 2014*, volume 8540 of *LNCS*, pages 451–470. Springer, 2015.

[LWP+25] Yongchao Li, Yongzhuang Wei, Enes Pasalic, Lang Li, and Ting Fan. LLBC: A novel Feistel-based low-latency block cipher for IoT applications. *IEEE Internet of Things Journal*, 12(21):45583–45595, 2025.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *EUROCRYPT 1993*, volume 765 of *LNCS*, pages 386–397. Springer, 1994.

[MP13] Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Paper 2013/328, 2013.

[ST17] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects: Revealing structural properties of several ciphers. In *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 185–215. Springer, 2017.