## Ultra Low-Latency Block Cipher uLBC

Guoxiao Liu<sup>1</sup>, Qingyuan Yu<sup>3,4</sup>, Liyuan Tang<sup>1</sup>, Shihe Ma<sup>1</sup>, Congming Wei<sup>5</sup>, Keting Jia<sup>a, 1,2,6</sup>, Lingyue Qin<sup>2,6</sup>, Xiaoyang Dong<sup>1,2,6</sup>, and Yantian Shen<sup>7</sup>

 $^1$ Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China $^2$ Zhongguancun Laboratory, Beijing, China

<sup>3</sup> School of Cyber Science and Technology, Shandong University, Qingdao, China

<sup>4</sup> Key Laboratory of Cryptologic Technology and Information Security, Jinan, China

<sup>5</sup> School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China
<sup>6</sup> BNRist, Tsinghua University, Beijing, China

<sup>7</sup> Department of Computer Science and Technology, Tsinghua University, Beijing, China

**Abstract.** In recent years, there has been a growing interest in low-latency ciphers. Since the first low-latency block cipher PRINCE was proposed at ASIACRYPT 2012, many low-latency primitives sprung up, such as Midori, MANTIS, QARMA and SPEEDY. Some ciphers, like SPEEDY and Orthros, introduce bit permutations to achieve reduced delay. However, this approach poses a challenge in evaluating the resistance against some cryptanalysis, especially differential and linear attacks. SPEEDY-7-192, was fully broken by Boura *et.al.* using differential attack, for example. In this paper, we manage to propose a novel low-latency block cipher, which guarantees security against differential and linear attacks. Revisiting the permutation technique used in Orthros, we investigate the selection of nibble permutations and propose a method for selecting them systematically rather than relying on random search. Our new nibble permutation method ensures the existence of impossible differential and differential trails for up to 8 rounds, while the nibble permutations for both branches of Orthros may lead to a 9-round impossible differential trail. Furthermore, we introduce a new approach for constructing low-latency coordinate functions for 4-bit S-boxes, which involves a more precise delay computation compared to traditional methods based solely on circuit depth. The new low-latency primitive uLBC we propose, is a family of 128-bit block ciphers, with three different versions of key length, respectively 128-bit and 256-bit key, as well as a 384-bit tweakey version with variable-length key. According to the key length, named uLBC-128, uLBC-256 and uLBC-384t. Our analysis shows that uLBC-128 exhibits lower latency and area requirements compared to ciphers such as QARMA<sub>9</sub>-128 and Midori128. On performance, uLBC-128 has excellent AT performance, the best performance except SPEEDY-6, and even the best performance in UMC 55nm in our experiments.

Keywords: Block Cipher Design · Low Latency Cipher · Low Latency S-box

## 1 Introduction

Block cipher is a fundamental cryptographic primitive, which is widely used to provide confidentiality in software and hardware systems. Designing a block cipher is a complex

<sup>a</sup>Corresponding author



E-mail: lgx22@mails.tsinghua.edu.cn (Guoxiao Liu), yuqy@mail.sdu.edu.cn (Qingyuan Yu), tangly22@mails.tsinghua.edu.cn (Liyuan Tang), msh21@mails.tsinghua.edu.cn (Shihe Ma), weicm@bi t.edu.cn (Congming Wei), ktjia@tsinghua.edu.cn (Keting Jia), qinly@tsinghua.edu.cn (Lingyue Qin), xiaoyangdong@tsinghua.edu.cn (Xiaoyang Dong), shenyt22@mails.tsinghua.edu.cn (Yantian Shen)

task that requires a carefully balanced strategy considering factors like cryptographic strength, latency, implementation cost, power, and energy consumption. The choice of trade-offs depends on the specific application requirements.

Low-latency ciphers have gained significant attention in recent years and offer notable advantages in high-throughput scenarios, including the Internet of Things (IoT) and encrypted hardware components of CPUs. In 2012, Borghoff *et al.* introduced the first low-latency block cipher named PRINCE [BCG<sup>+</sup>12], which proposes the  $\alpha$ -reflection structure where the encryption and decryption are similar to the keys K and  $K \oplus \alpha$ , respectively. The round function of PRINCE resembles the structure of AES, except that it incorporates  $4 \times 4$  almost MDS binary matrices with a branch number of 4, instead of the MDS matrix in the MixColumns mapping. Subsequently, a series of ciphers were introduced, such as Midori [BBI<sup>+</sup>15], MANTIS [BJK<sup>+</sup>16], QARMA [Ava17] and SPEEDY [LMMR21] etc.

Midori [BBI<sup>+</sup>15], introduced at ASIACRYPT 2015, is a family of two block ciphers: Midori64 and Midori128, each with a different block length. Midori128 applies an AES-like design, with the 8-bit S-boxes as the confusion module and  $4 \times 4$  almost MDS binary matrices as the diffusion module. The 8-bit S-boxes consist of two 4-bit S-boxes processed in parallel with bit permutation both in input and output, which can minimize the path delay in the round-based implementation. There are some cryptanalysis results that threaten the security of Midori. Guo et al. gave the invariant subspace attack against the full block cipher Midori64 [GJN<sup>+</sup>15]. Gérault *et al.* gave a practical attack on Midori64 and Midori128 under related key model [GL16]. MANTIS with 64-bit block length was introduced in CRYPTO 2016 [BJK<sup>+</sup>16], as a low-latency variant of the SKINNY family. MANTIS is a tweakable block cipher used for memory encryption, incorporating a TWEAKEY schedule [JNP14] derived from PRINCE. It also employs the involutory of Midori's S-box to optimize for small areas and low circuit depth. QARMA [Ava17] draws inspiration from PRINCE and MANTIS but employs a three-round Even-Mansour scheme rather than an FX-construction, which was improved to QARMA V2 with longer tweaks and improved security bounds [ABD<sup>+</sup>23]. In SAC 2020, the authors of PRINCE introduced PRINCE v2 [ $BEK^+21$ ], which improves the security of PRINCE without altering the number of rounds or rotation operations. Recently, some new low-latency block ciphers have been proposed. SPEEDY [LMMR21] incorporates CMOS hardware into the cipher design and applies a high-speed 6-bit S-box implemented using NAND gates. SPEEDY has a low delay for encryption without any requirements for area and decryption speed. However, Boura et al. give a differential attack for the full round SPEEDY-7-192, which breaks the expected security strength [BDBN23]. For some special application, Belkheyar et al. [BDD<sup>+</sup>23] designed a 24-bit low-latency tweakable block cipher with a 40-bit tweak, specifically for a memory safety concept called Cryptographic Capability Computing. In addition to the block ciphers, Banik et al. introduced a low-latency pseudorandom function named Orthros [BIL<sup>+</sup>21], which takes a 128-bit message and a 128-bit key as input and produces a 128-bit random number as output. Orthros is constructed using two 128-bit low-latency permutations. To minimize latency, the permutation involves 4-bit S-boxes, bit permutation, nibble permutation, and  $4 \times 4$  almost MDS (Maximum Distance Separable) binary matrices.

In the context of the low-latency block cipher, it is necessary to construct S-boxes that have low-latency. There are many methods for the selection of low-latency S-boxes [BGLS19, Ras22]. When constructing low-latency S-boxes, designers can use different metrics to measure the latency of coordinate functions. The most popular metric is depth, which assesses the path delay of the S-box by considering the sequential performance of the involved transistors. However, it treats NAND and NOR equally, which needs to be improved, just as analyzed in [LMMR21].

**Our contribution.** This paper proposes a new 128-bit block cipher while providing a strong security argument for its resistance against differential and linear attacks, which

has the lowest latency among the 128-bit block cipher as far as we know.

Among the low-latency ciphers, SPEEDY applies a low-latency 6-bit S-box, with a complex diffusion function, which makes it hard to give an accurate estimation of the probability of differential paths and linear paths. Midori128 and QARMA-128 both employ a byte-based design similar to AES, which facilitates establishing the lower bound for the active S-boxes. The round function utilizes a low-latency 8-bit S-box composed of two 4-bit S-boxes operating in parallel. The diffusion layer utilizes a selective byte permutation and an almost  $4 \times 4$  MDS matrix, with a branch number of 4. For the low-latency pseudorandom function (PRF) Orthros, which adopts a 128-bit permutation, a 32-nibble permutation, and the above almost MDS binary matrix. The use of bit permutation in the cipher makes it a challenge to find the upper bound of the optimal differential characteristic. Therefore, we specifically focus on the 32-nibble permutation for the security proof.

We use the 4-bit S-box, nibble permutation, and the almost  $4 \times 4$  MDS matrix to construct the block cipher. This transformation has a branch number of 4, allowing an active nibble to propagate to 27 nibbles at most after three rounds. Therefore, the aim is to have an active nibble that can diffuse to any of the 32 nibbles (a case commonly referred to as full diffusion). It is desired to have a round number of at least 4. In this case, the selection of a nibble permutation resistant to the differential attack is of great importance. We propose a well-chosen method for the nibble permutation instead of the random searching of the **Orthros** which can guarantee both the impossible differential and differential characteristic to be present 8 rounds at most.

4-bit S-boxes are employed to optimize the circuit's depth. A new method for constructing low-latency coordinate functions is proposed. The proposed method involves utilizing the logical effort metric to calculate the path delay of logic gates, combining this with effort delays and parasitic delays to design Boolean functions that achieve minimal latency while meeting specific cryptographic criteria. Utilizing the appropriate combination of coordinate functions, a large number of low-latency S-boxes are generated. Then we perform cryptographic analysis on these S-boxes and select one with the optimal security characteristics. We hope that our findings will inspire others to design new and efficient low-latency cryptographic primitives.

Besides, we give an optimal lightweight key schedule, which supports the extension of tweak block cipher following the TWEAKEY framework [JNP14]. We give a low-latency version of uLBC-128 and uLBC-256, which are given proof of resistance against differential and linear attacks. A secure tweakey cipher version with a 384-bit length of tweak and keys against related-key attacks is presented, which supports variable key lengths from 128-384 bits.

We give the full unrolled hardware implementation for ciphers Midori, QARMA, AES, etc. Among them, uLBC-128 is the lowest latency among the 128-bit block ciphers with the same security claim, On performance, uLBC-128 has excellent AT performance, the best performance except SPEEDY-6, and even the best performance in UMC 55nm in our experiments. Our implementations are publicly available in https://github.com/Guoxi aoLiu/uLBC.

## 2 Descriptions of Block Cipher uLBC

The uLBC is a low-latency block cipher built upon Substitution Permutation Networks (SPN) and includes proven security against differential and linear attacks. This section begins with an overview of uLBC and then introduces its encryption and decryption algorithms, followed by a description of the key schedule algorithms.

#### 2.1Overview

The uLBC block cipher operates with 128-bit blocks and supports key lengths of either 128 or 256 bits. Additionally, a tweakable variant with a 384-bit tweakey is available. These versions are designated as uLBC-128, uLBC-256, and uLBC-384t, respectively.

In the realm of block cipher security, particularly for applications requiring low latency, we emphasize SK-security, which ensures protection in a single-key setting where an attacker can access the encryption and decryption oracles under an unknown key. SK-security is a critical feature for practical security applications. Several low-latency ciphers, such as Midori, QARMA, PRINCE, and SPEEDY, maintain SK-security, which enables us to optimize uLBC for high-performance, SK-secure configurations. In such configurations, uLBC-128 and uLBC-256 provide 128-bit and 256-bit security, respectively, in a single-key setting. Furthermore, uLBC-384t functions as a tweakable block cipher with a 384-bit tweakey, composed of a secret key and a public tweak. The combined length of the key and tweak can vary up to 384 bits, with a minimum key length of 128 bits. uLBC-384t offers RK-security, which ensures robust protection in both single-key and related-key settings. In a related-key scenario, the attacker can access encryption and decryption oracles under multiple unknown keys that have known relationships.  $\mathsf{uLBC-384t}$  provides (384 - t)-bit security against related-key attacks, where t represents the number of bits in the tweak, constrained by  $0 \le t \le 256$ . Key parameters for each version are detailed in Table 1.

1	able 1: The p	barameters c	of ulbe family.	
Version	Block size	Key size	Tweakey size	Round
uLBC-128	128  bits	128  bits	-	18
uLBC-256	128  bits	256 bits	-	22

384 bits

30

Table 1. The parameters of ul BC family

The uLBC computation follows a big-endian format. The 128-bit block  $b_0, \ldots, b_{127}$  is divided into 32 nibbles,  $s_{00} = b_0 b_1 b_2 b_3$ ,  $s_{01} = b_4 b_5 b_6 b_7$ , ...,  $s_{31} = b_{124} b_{125} b_{126} b_{127}$ . The nibble is the basic unit referred to as a cell. The block includes 32 cells organized in a 4 by 8 matrix, as illustrated in the following.

(	$s_{00}$	$s_{04}$	$s_{08}$	$s_{12}$	$s_{16}$	$s_{20}$	$s_{24}$	$s_{28}$	)	
	$s_{01}$	$s_{05}$	$s_{09}$	$s_{13}$	$s_{17}$	$s_{21}$	$s_{25}$	$s_{29}$		
	$s_{02}$	$s_{06}$	$s_{10}$	$s_{14}$	$s_{18}$	$s_{22}$	$s_{26}$	$s_{30}$		
	$s_{03}$	$s_{07}$	$s_{11}$	$s_{15}$	$s_{19}$	$s_{23}$	$s_{27}$	$s_{31}$	Ϊ	

#### 2.2Encryption

uLBC-384t

128 bits

The block cipher  $\mathsf{uLBC}$  is based on the SPN structure. First, the plaintext state P is exclusive-ored with the whitening key  $RK_0$ , resulting in an intermediate state. This state then undergoes N rounds of a defined round function sequence, followed by N rounds of round functions. The round function includes S-box substitution (SubNib), constant addition (AddConst), cell permutation (PosPerm), column mixing (MixColumn), and round key addition (AddRK) operations. The round function is illustrated in Figure 1 and the encryption is given in Algorithm 1. The  $RK_1, RK_2, \ldots, RK_N$  are round keys generated by the key schedule algorithm. The final output is the ciphertext C. We provide test vectors of all versions of  $\mathsf{uLBC}$  in Appendix A.



Figure 1: The round function of uLBC.

Algorithm 1 The Encryption of uLBC.

<b>Input:</b> plaintext: $P$ , round key: $RK_0, \ldots, RK_N$	
Output: ciphertext: C	
$X_0 = P \oplus RK_0$	$\triangleright$ XORed with the white ning key
for $i = 0$ to $N - 2$ do	$\triangleright$ This is the <i>r</i> -th round $(r = i + 1)$
$Y_i = \mathbf{SubNib}(X_i)$	
$Z_i = \mathbf{AddConst}(Y_i)$	
$U_i = \mathbf{PosPerm}(Z_i)$	
$W_i = \mathbf{MixColumn}(U_i)$	
$X_{i+1} = W_i \oplus RK_{i+1}$	$\triangleright$ Round key addition
end for	
$Y_{N-1} = \mathbf{SubNib}(X_i)$	$\triangleright$ The last round
$X_N = Y_{N-1} \oplus RK_N$	
$C = X_N$	

**SubNib.** Substitution referred to as S-box is the only nonlinear operation based on nibbles, which divides 128-bit state X into 32 nibbles  $X = \{x_0, x_1, \ldots, x_{31}\}$ . These nibbles are substituted for other nibble values according to the table S-box, and the output state is obtained Y, where Y =**SubNib** $(X) = \{S(x_0), S(x_1), \ldots, S(x_{31})\}$ . The S-box is shown in Table 2.

Table 2: The (low-latency) S-D	$\sum S$
--------------------------------	----------

							· · ·									
Input	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
Output	0x8	0x0	0x1	0x5	0xc	0x7	0x4	0x6	0x2	0xa	0x3	0xd	0xe	0xf	0xb	0x9

Table 3: The first two nibbles of round constant addition.

Round	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$(c_0, c_1)$	0x80	0xc0	0xe0	0xf0	0xf8	0x7c	0xbc	0xdc	0xec	0xf4	0x78	0x3c	0x9c	0xcc	0xe4	0x70
Round	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$(c_0, c_1)$	0xb8	0x5c	0xac	0xd4	0x68	0x34	0x18	0xc0	0x84	0x40	0xa0	0xd0	0xe8	0x74	0x38	0x1c

Table 4: The last two by	ytes of round con	stant addition.
--------------------------	-------------------	-----------------

Round	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$(c_6, c_7)$	0xa0	0xac	0x93	0x29	0xac	0x4b	0xc9	0x91	0xc2	0x31	0x32	0x19	0xc1	0x93	0xca	0x81
Round	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$(c_6, c_7)$	0x44	0x20	0xcb	0x8b	0x49	0xcc	0x9b	0xa8	0x82	0xc1	0x04	0xba	0x4a	0x22	0xc9	0x18

**PosPerm.** A cell-wise permutation  $P_S$  is applied to the state.  $U = \text{PosPerm}(Z) = \{u_0, u_1, \ldots, u_{31}\}$ , where  $u_i = z_{P_S(i)}$ ,  $i = 0, 1, \ldots, 31$ . The permutation  $P_s$  is shown in Table 5.

 Table 5: The cell permutation  $P_S$ .

 Output  $P_S(x)$ 

Inj	put :	x							Out	put I	$P_S(x)$					
0	4	8	12	16	20	24	28	]	0	4	8	12	20	16	28	24
1	5	9	13	17	21	25	29	$\rightarrow$	25	1	29	9	5	13	17	21
2	6	10	14	18	22	26	30		18	30	22	26	10	2	6	14
3	7	11	15	19	23	27	31		15	19	7	23	31	27	3	11

**MixColumn.** Multiply each column of the intermediate state matrix with a  $4 \times 4$  binary matrix M, and the matrix M is shown below:

$$M = \left(\begin{array}{rrrrr} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{array}\right),$$

where the matrix  $M^{-1} = M$ .  $W = \{w_0, w_1, ..., w_{31}\} = MixColumn(U)$ , that is:

 $\begin{pmatrix} w_{4j} \\ w_{4j+1} \\ w_{4j+2} \\ w_{4j+3} \end{pmatrix} = M \begin{pmatrix} u_{4j} \\ u_{4j+1} \\ u_{4j+2} \\ u_{4j+3} \end{pmatrix}, \quad j = 0, 1, \dots, 7.$ 

If the cells are not all equal to 0 for the input  $\{u_{4j}, u_{4j+1}, u_{4j+2}, u_{4j+3}\}$ , there are at least four non-zero cells among the input and output cells. Hence, the branch number is 4.

AddRK. The state W after the column mixing operation and the round key  $RK_r$  (r = 1, 2, ..., N) are bitwise exclusive-ored to obtain the output state.

#### 2.3 Decryption

The decryption algorithm is the inverse operation of the encryption algorithm, involving the execution of N-round iterative operations. These operations include the addition of the round key, the inverse of column mixing (MixColumn<sup>-1</sup>), the inverse of cell permutation (PosPerm<sup>-1</sup>), and the inverse of the S-box substitute (SubNib<sup>-1</sup>). The decryption ends with an exclusive OR operation with the key  $RK_0$ , producing the plaintext. The pseudocode for the decryption algorithm is presented in Algorithm 2.

AddRK. During decryption, the initial state  $X_N$  is ciphertext C. In the decryption process of the *r*-th round (r = 1, 2, ..., N), the state  $X_{N-r+1}$  and the round key  $RK_{N-r+1}$  are exclusive-ored, and the last state  $X_0$  and key  $RK_0$  are exclusive-ored to obtain the output plaintext P.

**MixColumn**<sup>-1</sup> Same as **MixColumn**, because of  $M^{-1} = M$ , multiply each column of the intermediate state matrix with the  $4 \times 4$  binary matrix M.

 $PosPerm^{-1}$  In the state of 32 nibbles, cell permutation is performed. The cell permutation position of these 32 units is in Table 6.

Algorithm 2 The Decryption of uLBC.

**Input:** ciphertext: C, round key:  $RK_0, \ldots, RK_N$ Output: plaintext: P  $X_N = C$  $Y_{N-1} = X_N \oplus RK_N$  $\triangleright$  The first round  $X_{N-1} = \mathbf{SubNib}^{-1}(Y_{N-1})$ for i = N - 2 to 0 do  $\triangleright$  This is the *r*-th round (r = N - i) $W_i = X_{i+1} \oplus RK_{i+1}$  $\triangleright$  Round key addition  $U_i = \mathbf{MixColumn}^{-1}(W_i)$  $Z_i = \mathbf{PosPerm}^{-1}(U_i)$  $Y_i = \mathbf{AddConst}(Z_i)$  $X_i = \mathbf{SubNib}^{-1}(Y_i)$ end for  $P = X_0 \oplus RK_0$  $\triangleright$  XORed with the whitening key

Table 6: The cell permutation  $P_S^{-1}$ .

In	put :	x							Out	put 1	$P_S^{-1}(x)$	;)				
0	4	8	12	16	20	24	28		0	4	8	12	20	16	28	24
1	5	9	13	17	21	25	29	$\rightarrow$	5	17	13	21	25	29	1	9
2	6	10	14	18	22	26	30		22	26	18	30	2	10	14	6
3	7	11	15	19	23	27	31		27	11	31	3	7	15	23	19

AddConst The first two columns of the state and 32-bit constant  $c = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$  are XORed, and the constant value is in the opposite order to the encryption. SubNib<sup>-1</sup> The S-box substitution inversion is also a non-linear operation performed on nibbles. The 32 nibbles are substituted one by one through the inverse S-box table to obtain the output state X =SubNib<sup>-1</sup>(Y).

#### 2.4 Key Schedule

The key scheduling algorithm incorporates cell permutation to reduce area and improve efficiency. The key schedule of uLBC-256 exhibits slightly greater complexity compared to that of uLBC-128. We use the functions  $f_1$  and  $f_2$  for the nibble, which are based on LFSR construction with a cycle of 15.

For uLBC-128, the key length equals the block length. Its key schedule algorithm uses the function F and key K to generate round keys  $RK_i$ , i = 0, 1, ..., N, that is,  $RK_0 = K$ ,  $RK_i = F(RK_{i-1})$ , i = 1, ..., N.

The function F is a cell permutation. The 128-bit state is divided into 32 nibbles (4-bit cells). The nibble-based matrix has four rows and eight columns. The permutation of the cell position is shown in Table 7. The input and output of F function are  $X = \{x_0, x_1, \ldots, x_{31}\}$  and  $Y = \{y_0, y_1, \ldots, y_{31}\}$  respectively.

The pseudocode of the round function Y = F(X) is  $y_i = x_{P_K[i]}, i = 0, 1, \dots, 31$ .

					Tabl	e 1. (	en b	ermu	latio	n tai	ne i	ĸ٠				
Inj	put a	į							Out	put I	$P_K[i]$					
0	4	8	12	16	20	24	28		8	9	20	18	4	10	21	27
1	5	9	13	17	21	25	29	$\rightarrow$	24	31	19	2	15	29	16	13
2	6	10	14	18	22	26	30		28	14	26	6	11	12	7	22
3	7	11	15	19	23	27	31		23	17	5	30	25	3	0	1

Table 7: cell permutation table  $P_K$ 

For uLBC-256, the key length is twice the block length. The key scheduling algorithm initially divides the 2*n*-bit master key K into two *n*-bit round keys  $K = K_0 || K_1$ . Subsequently, it employs the F function and the  $f_1$  function to generate the round keys. The

 $f_1$  function involves nibble transformation, while the F function remains consistent with uLBC-128. Specifically,  $RK_0 = K_0 \oplus K_1$ ,  $RK_i = F^i(K_0 \oplus f_1^i(K_1))$ ,  $i = 1, \ldots, N-1$ .  $f_1$  is constructed on the basis of a 4-bit LFSR with a feedback polynomial  $(a_0, a_1, a_2, a_3) \rightarrow (a_0 \oplus a_3, a_0, a_1, a_2)$ . The corresponding inputs and outputs are detailed in Table 8.

	Table 8: Nibble substitution table for $f_1$ .															
Input	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
Output	0x0	0x8	0x1	0x9	0x2	0xa	0x3	0xb	0xc	0x4	0xd	0x5	0xe	0x6	0xf	0x7

For uLBC-384t, we regard the key and tweak as a unified tweakey, following the TWEAKEY(STK) construction in [JNP14]. The length of key and tweak can be flexibly chosen, but the length of key must be at least as large as the block size, i.e., 128-bit. The tweakey schedule also first divides the 3n (z = 3 as [JNP14]) length tweakey TK into three n-bit round tweakeys, as  $TK = TK_0||TK_1||TK_2$ . Subsequently, the F function is applied as a cell permutation, similar to uLBC-128 in each round of key schedule, and the  $f_1$  and  $f_2$  functions are utilized for the cell transformations. The specifics are detailed below:

$$RK_0 = TK_0 \oplus TK_1, \ RK_i = F^i(TK_0 \oplus f_1^i(TK_1) \oplus f_2^i(TK_2)), \ i = 1, \dots, N-1$$

The  $f_1$  and  $f_2$  functions are based on lightweight 4-bit LFSR, which should be carefully chosen to ensure security in related-key (related-tweak) models. As stated in [QDW<sup>+</sup>22], applying the LFSR to a nibble is equivalent to multiply the nibble by a  $4 \times 4$  binary matrix. When  $f_1$  is corresponding to L, we choose the binary matrix of  $f_2$  as  $L^{-1}$ , where

$$L = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, L^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}.$$

So the  $f_1$  and  $f_2$  functions satisfy the subtweakey difference cancellation property to STK construction: for a given cell position, z - 1 cancellations can only happen every 15 rounds, which can be proved as [QDW<sup>+</sup>22]. The feedback polynomial of  $f_2$  is  $(a_0, a_1, a_2, a_3) \rightarrow (a_1, a_2, a_3, a_0 \oplus a_1)$ , and the corresponding inputs and outputs of  $f_2$  are given in Table 9.

Table 9: Nibble substitution table for  $f_2$ .

Input	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
Output	0x0	0x2	0x4	0x6	0x9	0xb	0xd	0xf	0x1	0x3	0x5	0x7	0x8	0xa	0xc	0xe

## **3** Design Decisions

#### 3.1 Main Ideas and Strategies of Design

This paper aims to design a family of low-latency block ciphers while providing strong security arguments. AES is the most widely deployed block cipher, known for its bytesbased design framework with high throughput and strong security. AES, with a strong diffusion layer comprising ShiftRows and MixColumns, has been proven resistant to various attacks such as differential, impossible differential, and linear attacks. In AES, the MixColumns operation employs an MDS matrix with a branch number of 5. To meet minimal area and energy requirements, the block cipher Midori128 utilizes an AES-like design. It incorporates a low-latency 8-bit S-box composed of two 4-bit S-boxes. Instead of using ShiftRows, it utilizes a byte permutation, and it replaces the original MDS matrix of AES with an almost MDS matrix having a branch number of 4. To obtain low-latency, the PRF of Orthros XORs two 128-bit outputs of permutations, each permutation employing bit permutation and nibble permutations, along with a  $4 \times 4$  almost MDS matrix, to facilitate strong diffusion. However, Orthros is solely used as a pseudo-random function. The cipher employs bit permutation, making it challenging to estimate the lower bound of the active S-box count to derive the security arguments.

In the context of proving the security of a low-latency block cipher, we focus on components such as 4-bit S-boxes (or 8-bit S-boxes), 32-nibble permutations (or 16-byte permutations), and low-latency MDS matrices (or almost MDS matrices). Among these options, we identify a round function with a 4-bit S-box as the confusion function, a nibble permutation, and an almost MDS matrix as the diffusion function. This combination achieves low latency and low area, making it suitable for efficient hardware implementations. The choice of a 4-bit S-box is friendly for AVX instructions to enhance software implementations using parallel modes, such as CTR and GCM.

The selection of nibble permutation in the diffusion function has been optimized within the bounds of the number of active S-boxes and the length of an impossible differential. We carefully choose the nibble permutation to enhance the bounds on the number of active S-boxes. While several design elements are inspired by existing ciphers, we introduce new methods aimed at achieving low-latency and lightweight performance.

Furthermore, we apply 4-bit S-boxes to optimize the depth of the circuit. A new method for constructing low-latency coordinate functions is proposed, which is more consistent with the implementation. Using a proper combination of coordinate functions, a large number of low-latency S-boxes are constructed.

Based on the above research, we propose a 128-bit block cipher that supports both 128bit and 256-bit keys. For lightweight design, we apply a linear key schedule based on nibble permutation for 128-bit keys. This schedule effectively generates key diffusion, providing resistance against related-key attacks and weak-key attacks. The nibble permutation is employed to extend the TWEAKEY frame of SKINNY[BJK<sup>+</sup>16] for the 256-bit key configuration. The Tweakey frame can support longer tweaks and keys. Therefore, we present a tweakable block cipher with 384-bit tweakey. The combined length of the key and tweak is flexible at 384 bits, yet the key must be at least 128 bits. The Tweakey framework treats the key and tweak interchangeably. Hence, it is necessary to provide related-key attack resistance in Tweakey versions.

#### 3.2 The Design of Round Function

We employ a 128-bit block size, which is a commonly used size to optimize encryption and decryption efficiency. Our cipher utilizes the Substitution-Permutation Network (SPN) structure, with the round function consisting of both a confusion layer and a diffusion layer. In this section, we focus on selecting the appropriate components. The 4-bit S-boxes are widely used in low-latency ciphers such as Midori, QARMA, MINTIS, etc.

Here, We take 4-bit S-boxes as the basic cell to design a 128-bit block cipher, which means that the 128-bit block is divided into 32 4-bit S-boxes. In addition, a linear layer is constructed for the 32 nibbles, which are represented by a  $4 \times 8$  matrix. Here, the nibble permutation plays an important role in determining the number of active S-boxes. Here, we use a reliable nibble permutation **PosPerm** to compare the differential diffusion of the MDS matrix and the almost MDS matrix. For the diffusion layer, we compare the efficiency of the following two cases:

**Case I** We use nibble permutation (**PosPerm**) and almost MDS matrix M, which is shown in Subsection 2.2.

**Case II** We use nibble permutation (**PosPerm**) and the low-latency MDS matrix  $M_4$ . The modular polynomial is  $x^4 + x + 1$ , with values 0x13. This MDS is given in the [LS16]. There are 4 XOR operations for each output bit.

$$M_4 = \begin{pmatrix} 1 & 1 & 4 & 9 \\ 1 & 4 & 9 & 1 \\ 4 & 9 & 1 & 1 \\ 9 & 1 & 1 & 4 \end{pmatrix},$$

For cryptanalysis, the number of 4-bit active S-boxes is the main criterion for evaluation against differential and linear attacks. We analyze the minimized number of active S-boxes in cryptanalysis and the rounds of the full diffusion, which are listed in Table 10. Midori128 incorporates a low-latency 8-bit S-box composed of two 4-bit S-boxes which are the basic units. It has 41 active S-boxes for 9-round differential characteristics, 62 active S-boxes for 12-round differential characteristics, and 67 active S-boxes for 13-round differential characteristics. However, there are 68 active S-boxes for 9-round differential characteristics in Case I and 74 active S-boxes for 9-round differential characteristics in Case II, both more than 64 active S-boxes. Hence, there are at most 8-round differential characteristics for the two cases. It is evident that the latter two have stronger diffusion to resist differential attacks.

Now we consider the low latency of the hardware implementation. The diffusion layer has the same XOR computation for Midori128 and **Case I**. Here we compare the implementation for **Case I** and **Case II**. There are 4-round full diffusion for **Case I** and 3-round full diffusion for **Case II**. We compare the latency for 18-round **Case I** and 16-round **Case II**, and the first case has a little advantage over the second case. Hence, we use the round function of **Case I**. Subsequently, we introduce the design of the diffusion layer and the low-latency S-boxes to improve the throughput.

Table 10: The number of active S-boxes of differential for the 32-nibble block.

lincon lovon	round number					rour	ds			
imear layer	of full diffusion	1	<b>2</b>	3	4	5	6	7	8	9
Midori128	3	1	4	7	16	23	30	35	38	41
Case I	4	1	4	$\overline{7}$	16	25	40	52	60	68
Case II	3	1	5	9	25	41	55	58	62	74

#### 3.3 The Design of Diffusion Layer

We use Case I as the round function, which is composed of a nibble permutation and a  $4 \times 4$  almost MDS binary matrices. The almost MDS binary matrices have been used in the design of PRINCE, Midori, MANTIS, Orthros, etc. However, the  $4 \times 4$  almost MDS binary matrices have a branch number of 4 less than the MDS matrices, resulting in slow diffusion. Banik *et al.* utilize bit and nibble permutations in a hybrid manner to improve the diffusion speed and to increase active S-boxes in each round [BIL<sup>+</sup>21]. However, the bit permutation is not suitable for software implementation, which usually makes the implementation speed slow. Besides, the search space for bit permutation is too large to give accurate estimates of the active S-box number in the differential or linear path. Here, we applied the nibble permutation. There are  $32! \approx 2^{117}$  permutations for 32 nibbles. It is impossible to traverse. Banik *et al.* give some analysis to select good nibble permutation, such as Condition 1.

**Condition 1** (Condition 3 [BIL<sup>+</sup>21]). For each column  $(u_{4i}, u_{4i+1}, u_{4i+2}, u_{4i+3})$  in the  $4 \times 8$  array, after applying the nibble permutation, they will be mapped to four nibble-cells in different columns.

Banik *et al.* randomly choose 7,000 nibble permutations satisfying condition 1, and then compute the lower bound of the number of active S-boxes after 5, 6, 7, and 8 rounds

for these nibble permutations with the MILP model. Among them, they find three nibble permutations that can achieve 60 active S-boxes over 8 rounds.

The MILP model is a time-consuming operation. Here we give some more conditions to find a good nibble permutation efficiently.

**Condition 2.** For each column  $(u_{4i}, u_{4i+1}, u_{4i+2}, u_{4i+3})$  in the  $4 \times 8$  array, after applying the nibble permutation twice, they will be mapped to four nibble-cells in different columns.

This condition is used to optimize the bound of active boxes in the differential path.

**Condition 3.** For each active input of the S box in the first round, it will generate 9 active S-box after two rounds, which will be mapped to all 8 columns after the **PosPerm**.

In the case of an active box transformation,  $1 \rightarrow 3 \rightarrow 9 \rightarrow (18 \sim 27) \rightarrow 32$ . Hence, the 4-round diffusion needs to be taken into account. We give the following conditions.

**Condition 4.** For each active S-box input in the first round, it will propagate the difference to all cells in the fourth round, i.e., there are at least 2 active S-box in each column before **MixColumn** in the fourth round.

The Condition 4 is used to ensure the full diffusion, which is used to limit the length of an impossible differential. For the selection of the nibble permutation, we specifically limit impossible differential distinguishers to a maximum of 8 rounds.

Because **MixColumn** can mix the column, we independently choose the row permutation for each row satisfying all the above conditions. There are exactly 8! = 40320 row permutations. We searched all these cases, and there are about 695520 permutations inconsistent with all 4 conditions. Then we search the differential path by estimating the active S-boxes for 6 to 9 rounds. We found that when the number of active S-boxes is 40 for 6-round uLBC with a selected nibble permutation, it is hard to find a 9-round differential with high probability. The permutation found is row-independent, therefore, we can implement the round function with the AVX instructions efficiently.

Table 11 shows a comparison of the lower bound of the number of active S-boxes for Midori128, Branch1 and Branch2 of Orthros, and uLBC-128. Compared with Branch1/2 of Orthros, our nibble permutations guarantee a much larger number of active S-boxes for 6 rounds and 7 rounds. Our nibble permutation causes a stronger diffusion at the same time, which leads to an 8-round impossible differential. However, we find many 9-round impossible differentials for Branch1/2 of Orthros, respectively, seen in Table 12 and Table 13, where '\*' means the none zero difference nibble, '0' means the zero difference is unknown, and  $a \neq 0$ .

Table 11: Comparison of lower bounds of the number of active S-boxes.

	nun	nber o	of act	rounds of IMP			
rounds	4	5	6	7	8	9	
Midori128 [BBI+15]	16	20	30	35	38	41	7 [TAY17]
Orthros Branch1 [BDD <sup>+</sup> 23]	16	25	36	50	60	67+	9 (Table 12)
Orthros Branch2 [BDD <sup>+</sup> 23]	16	25	36	51	60	67+	9 (Table 13)
Our nibble permutation		25	40	52	60	68	8 (Table 18)

Position	State
$\Delta_f$	000000000000000000000000000000000000000
Round 1	00000000000*0**000000000000000000
Round 2	0000000***000000***0000000**0*
Round 3	0***0000*0*****0?**?*0**?**?***0
Round 4	????????***0???????????????????????????
Round 4	????*??????*??????*????*???*???*??
Round 5	?*0***0?****0**000?*0*?***00****
Round 6	0*0000*000*00**0*0000*0*0000
Round 7	00*0*0000*00000000000000000000000000000
Round 8	000000000000000000000000000000000000000
$\Delta_r$	00000000000a0aa00000000000000000000000

Table 12: An impossible differential trail of Branch1.

Table 13: An impossible differential trail of Branch2.

	I construction of the second s
Position	State
$\Delta_f$	00000000000000*000000000000000000000000
Round1	000000000000000000000000000000000000000
Round2	0000**0***0*0000*0**000000000000
Round3	??**0000?**?0000*0**?**?***00***
Round4	???*????????*?*????????***0????
Round4	???????????????????????????????????????
Round5	0**?*00**0*?0??0**0*?*?0*0*0***0
Round6	*0*00*00000*0000**0000**00
Round7	0000000*000*0*0000000000000000000000000
Round8	*00000000000000000000000000000000000000
$\Delta_r$	00000000000000000000000000000000000000

#### 3.4 The Design of 4-bit S-boxes

The 4-bit S-box is widely used in symmetric primitives due to its favorable cryptographic properties, compact area footprint, low latency, and relatively low cost for increasing side-channel protection. The block cipher uLBC also employs a 4-bit S-box, which exhibits very low latency and incorporates various security features such as differential probability, linear bias advantage, algebraic degree, fixed points, and more. Here are the security indicators of the S-box used in the uLBC algorithm: 1) The maximum difference probability is  $2^{-2}$ . 2) The maximum linear approximation probability is  $2^{-2}$ . 3) The highest algebraic degree of the S-box (and the inverse) is 3. 4) The S-box is a permutation without any fixed points.

#### 3.4.1 Logic Gates and Their Latency

To evaluate the latency of the coordinate functions, the most popular metric is depth, which assesses the path delay of the S-box by considering the sequential performance of the logic gates involved [BBI<sup>+</sup>15, BGLS19, Ras22]. Leander *et al.* highlighted the limitations of the depth metric by analyzing the latency of logic gates by calculating the Fan-in-to-Latency Ratio (FLR) for each gate and observing that NAND and OAI gates get the highest FLR scores among all logic cells tested [LMMR21].

When building 4-bit S-boxes, we found that NAND does have the lowest latency but some logic gates also perform well in terms of latency, which is ignored in [LMMR21]. The traditional depth metric is based on the number of logic gates to sequentially proceed in the operation and only considers gates with fan-in of no more than 2.

In this paper, we introduce *logical effort* as the metric instead of depth and give logical efforts for some general logic gates. The logical effort method, introduced by Sutherland

et al. [SSH99], offers a technique to design MOS circuits to achieve high speeds. This approach simplifies MOS circuits as networks of resistance and capacitance for circuit analysis. The delay of a logic gate d is the summation of the parasitic delay p and the effort delay f, defined as d = f + p. The effort delay f is the product of the logical effort g and the electrical effort h, given by f = gh. The electrical effort h characterizes the cost of driving capacitive loads, commonly referred to as fanout by many CMOS designers. For simplicity, we assume it is fixed to 1 in the subsequent discussion. Logical effort qdescribes the computational cost attributable to the circuit topology. The logical effort of the inverter is defined as 1 delay unit, thus all logical efforts are gauged relative to the delay of a basic inverter. Since the logical effort of a gate can be computed by assessing how much additional input capacitance a gate presents to deliver the same output current as an inverter, the logical efforts of other gates are derived based on the ratio of input capacitances. The parasitic delay p of a logic gate is predetermined, with the primary contribution being the capacitance of the source/drain regions of the transistors that drive the gate output. The parasitic delay is a multiple of the parasitic delay of the inverter, which is conventionally set to 1 delay unit. Table 14 presents assumed values of the logical effort and the parasitic delay for various gates. This table was simplified based on [SSH99]. As [SSH99] is used mainly for circuit design, it takes into account various factors more carefully, such as the influence of fanouts, etc. Therefore, simplification is reasonable.

Table 14: Values of delay factors for some logical gates.

Gate	Fan-in	Logical effort $(g)$	Parasitic delay $(p)$
INV	1	1.0	1.0
NAND2	2	1.33	2.0
NOR2	2	1.67	2.0
NAND3	3	1.67	3.0
NOR3	3	2.33	3.0
OAI21	3	2.0	3.0
AOI21	3	2.0	3.0
NAND4	4	2.0	4.0
NOR4	4	3.0	4.0
OAI22	4	2.0	4.0
A0122	4	2.0	4.0

In logic networks, the path logical effort G is the product of the logical effort of each logic gate along the path, defined as  $G = \prod g_i$ . Further, the path effort F is computed as the product of G and H, i.e.,  $F = G \cdot H$ . Here,  $H = \prod h_i$  represents the effort of the individual stages along the path. In practice, it is often possible to expand the circuit and utilize additional area to decrease the number of fan-ins and fan-outs, resulting in H = 1. Consequently, the path effort simplifies to F = G, as the individual stage efforts cancel out. The path parasitic latency denoted as P, is defined as the sum of the parasitic latencies  $P = \sum p_i$ . For an N-stage logic network, the path delay can be computed by

$$\mathcal{D} = N \cdot F^{\frac{1}{N}} + P. \tag{1}$$

The path delay of Boolean functions is the maximum path delay of each input wire. We give an example to show how to compute the path delay.

**Example 1.** The path delay of a 3-bit S-box. The S-box is

$$\begin{split} y_1 &= \texttt{INV}(\texttt{NAND2}(\texttt{NAND2}(x_1, x_2), \texttt{NAND2}(\overline{x_1}, x_3))), \\ y_2 &= \texttt{NAND2}(\texttt{NAND2}(x_2, x_1), \texttt{NAND2}(\overline{x_2}, x_1)), \\ y_3 &= \texttt{NAND2}(\texttt{NAND2}(x_3, x_1), \texttt{NAND2}(\overline{x_3}, x_2)). \end{split}$$

For a 3-bit S-box, there are 3 input values (i.e.,  $x_1, x_2$ , and  $x_3$ ) and 12 input wires. We number them in sequence from 1 to 12, where the delays of index 1, 2, and 4 are the same, the delays of index 5, 6, 8, 9, 10, and 12 are the same, and the delays of index 7 and 11 are the same. Thus we can obtain

$$\mathcal{D}_{1,2,4} = 3 \cdot (1 \cdot 1.33 \cdot 1.33)^{1/3} + (1+2+2) = 8.63,$$
  
$$\mathcal{D}_3 = 4 \cdot (1 \cdot 1.33 \cdot 1.33 \cdot 1)^{1/4} + (1+2+2+1) = 10.61,$$
  
$$\mathcal{D}_{5,6,8,9,10,12} = 2 \cdot (1.33 \cdot 1.33)^{1/2} + (2+2) = 6.66,$$
  
$$\mathcal{D}_{7,11} = 3 \cdot (1.33 \cdot 1.33 \cdot 1)^{1/3} + (2+2+1) = 8.63.$$

Thus the path delay of this S-box is the maximum path delay among all Boolean functions, i.e.,  $\mathcal{D} = \mathcal{D}_3 = 10.61$ .

#### 3.4.2 S-box Searching

Using the path delay, the first step is to find several coordinate functions for 4-bit S-boxes with low latency. We select some common logic gates, as shown in Table 14, and construct Boolean functions based on the delays of all the logic gates along the path of the Boolean function. Table 14 lists values of delay factors for all the logic gates we selected. According to equation (1), the delay of a Boolean function  $\mathcal{D}$  is derived as  $\mathcal{D} = N \cdot F^{\frac{1}{N}} + P$ .

In our search, to address the challenge of path delays, we iterate based on the depths of logic gates. We assign weights to the depths of different gates, assuming that the depths of INV, NAND2/NOR2, NAND3/NOR3/OAI21/AOI21, and NAND4/NOR4/OAI22/AOI22 gates are weighted as 0.5, 1, 2 and 2, respectively. The BUF gate  $y = x_0$ ,  $y = x_1$ ,  $y = x_2$ ,  $y = x_3$ , and the INV gate  $y = \neg x_0$ ,  $y = \neg x_1$ ,  $y = \neg x_2$ ,  $y = \neg x_3$ , are set to have factors:

	Depth	Path logical effort	Path parasitic delay
BUF	0	1.0	0
INV	0.5	1.0	1.0

When searching Boolean functions whose depths are s, we apply INV, NAND2/NOR2, NAND3/NOR3/OAI21/AOI21, and NAND4/NOR4/OAI22/AOI22 to the Boolean functions with depth s - 0.5, s - 1, s - 2 and s - 2 respectively, and check whether the Boolean functions satisfy our requirements: 1) The path delay is no more than 10. 2) The depth is no more than 4.5. 3) The output is 0-1 balanced. 4) The function expression contains all 4 bits. 5) The algebraic degree is greater than 1. There is no special reason for the first and second conditions since some Boolean functions can already be found within these parameters. We can also set them based on the path delay of some known S-boxes, such as the SPEEDY S-box [LMMR21]. After that, we use these Boolean functions to construct 4-bit S-boxes by the method of [Ras22]. Note that to reduce the search space, we require that the coordinate functions of the S-box be sorted by truth table during the search. Our selected S-boxes are listed below:

Table 15: The structure of our low-latency S-box.  $S_i$  denotes the *i*th coordinate functions.

Function	Structure	Delay $\mathcal{D}$
$S_0$	$\texttt{NAND2}(\texttt{NAND3}(\texttt{INV}(x_0),\texttt{INV}(x_2),\texttt{INV}(x_3)),\texttt{OAI21}(x_1,x_3,x_0))$	9.8
$S_1$	$\texttt{NAND2}(\texttt{NAND3}(\texttt{INV}(x_0), x_1, x_3), \texttt{OAI21}(\texttt{INV}(x_2), \texttt{INV}(x_3), x_0))$	10.1
$S_2$	$\texttt{NAND2}(\texttt{NAND3}(\texttt{INV}(x_2), x_3, x_1), \texttt{OAI21}(x_0, \texttt{INV}(x_1), x_2))$	10.1
$S_3$	$\texttt{NAND2}(\texttt{NAND3}(\texttt{INV}(x_1), x_3, x_2), \texttt{OAI21}(\texttt{INV}(x_0), \texttt{INV}(x_2), x_1))$	10.1

We find a few S-Boxes that fit the conditions and come up with the one that performs best in Table 15. The DDT and LAT of the S-box S we used are shown in Figure 2.

Table 16 lists the depths, delays, and practical latency of S-boxes in some low-latency block ciphers. Our metric uses finer granularity than depth, for example, the S-boxes with depth 3.5 in Table 16 are further divided into two categories  $\mathcal{D} = 10$  and  $\mathcal{D} = 11$ , and the S-boxes with depth 4 are further divided into categories  $\mathcal{D} = 12$  and  $\mathcal{D} = 13$ . We observed that the latency of the S-boxes is basically in line with our estimates. Generally, S-boxes with high  $\mathcal{D}$  values take an advantage in latency, while there are a few exceptions, such as Midori-Sb1, which has lower latency than those S-boxes with  $\mathcal{D} = 11$ . Besides, in the experiments, the S-box of QARMA- $\sigma_2$  has lower latency than the inverse operation (QARMA- $\sigma_2$ \_Inv), but the depth value of QARMA- $\sigma_2$  is greater than that of QARMA- $\sigma_2$ \_Inv and did not match the measured values. When using our metric, both S-boxes have the same  $\mathcal{D}$  values, which is at least consistent with the experimental results of 45nm.

<u> </u>		•	Latency				
S-box	Depth	D	$15 \mathrm{nm}(\mathrm{ps})$	45nm(ns)			
Orthros	3.5	10	8.86985	0.12642			
Orthros_Inv	4	13	13.31488	0.19144			
Midori-Sb0	3.5	10	9.78337	0.11200			
Midori-Sb1	4	12	9.04462	0.12365			
$QARMA$ - $\sigma_0$	3.5	11	8.84451	0.11233			
$QARMA$ - $\sigma_1$	4	12	11.82724	0.14175			
$QARMA$ - $\sigma_2$	4.5	13	14.85347	0.17133			
$QARMA$ - $\sigma_2$ _Inv	4	13	12.48697	0.17870			
Our S-box	3.5	10	8.01972	0.12702			
Our S-box_Inv	4	11	14.13926	0.14475			

Table 16: Properties of some low latency S-boxes.

#### 3.5 The Design of Key Schedule and Constants

Block cipher uLBC supports a 128-bit block, and the key length supports 128 bits and 256 bits. We also design a version of the 128-bit block and 384-bit tweakey. Although a little complex key schedule does not increase encryption latency, we apply a simple key schedule by nibble permutation for the 128-bit key to reduce the cost of implementation.

For a 256-bit key, we make use of the nibble permutation to extend the Tweakey frame of SKINNY [JNP14] with tweaked size of twice the block length. The 256-bit key is divided into two equal parts. The nibble permutation F is used for the two parts independently. A cell transformation function  $f_1$  is used in each nibble of the second part for each round.  $f_1$ is constructed by a 4-bit LFSR with a period of 15. Hence, the masterkey can be computed by any two consecutive round subkeys. For the 384-bit tweakey, the key schedule is similar to the 256-bit key. The only difference is the 384-bit tweakey is divided into three parts, and the cell transformation functions  $f_1$  and  $f_2$  are used in each nibble of the last two parts for each round. The  $f_1$  and  $f_2$  functions adhere to the subtweakey difference cancellation property to STK construction given in [QDW<sup>+</sup>22]. And, the masterkey can be computed by any three consecutive round subkeys.

For the nibble permutation F in uLBC-128, uLBC-256, and uLBC-384t, we choose the same nibble permutation to reduce implementation costs. Hence, we add some conditions on the nibble permutation in the key schedule to remove some iteration paths in the related-key setting to sieve good nibble permutation in the following. (1) After the nibble permutation, the contents of a column in the key schedule are moved to four different columns. (2) The same column, after the key nibble permutation F, is positioned in a different column than after **PosPerm** permutation in the round function. (3) There are at least four active boxes for the 4-round related-key differential for uLBC-384. Then we check the nibble permutation by the number of active S-boxes in the related-key setting.

Many lightweight block ciphers apply a simple key schedule where round keys only differ by the addition of a round-specific constant, which is vulnerable to invariant subspace attacks. Beierle *et al.* [BCLR17] analyzed the resistance of some ciphers against invariant attacks and gave a method to find round constants that guarantee the resistance to all types of invariant attacks. Following that, we choose the round constants. There are three parts: one is used to distinguish four versions of uLBC; another part is constructed by the LFSR with a period of 63 to resist the invariant attacks; the last part is used as random numbers.

## 4 Security Analysis

This section provides cryptanalysis of the block cipher uLBC, including various attack techniques such as differential attack, boomerang attack, linear analysis, impossible differential analysis, integral analysis, and related-key analysis. The summary of the analysis results is shown in Table 19.

#### 4.1 Differential/Linear Cryptanalysis

#### 4.1.1 Basic Properties and Nibble-Level Bounds

The differential distribution table (DDT) and linear approximation table (LAT) of the S-box are shown in Figure 2. As we can see, the highest differential/linear probability for each S-box is  $2^{-2}$ .



Figure 2: Differential/Linear properties of the S-Box S.

To argue for the resistance of uLBC against differential and linear attacks, we computed lower bounds on the minimal number of active S-boxes. Based on bit-level differential propagation, we consider all possible differential characters of S-box and build an SAT automatic search model for solving the minimum S-boxes. We get the results for uLBC-128 with 1-9 rounds, the results are shown in Table 17. Because of the equivalence between differential and linear trails, the result for the linear attack is the same. For verification, we present our differential trails with 3 to 7 rounds in Table 22, Appendix B.

Table 17: Estimation of the number of active S-boxes
--

Round	1	2	3	4	5	6	7	8	9
Number	1	4	7	16	25	40	52	60	68

We prove that the minimum number of active S-boxes in 9-round uLBC-128 is at least 68, and the maximum differential characteristics probability will not be higher than  $2^{-136}$ .

Meanwhile, we consider the probability of clustering of differential trails. We search for clusterings with our SAT automatic search tools, the searching phase is carried out in two steps. First, we search for the differential trail with the maximum probability. Then, we fix the input and output differentials equal to the trail, and add constraints to ensure that the probability is greater than  $2^{-10}$  times of the maximum probability, while searching for all eligible trails. We continue this process until we have an adequate number of trails in the clustering. Finally, we evaluate the actual probability for 8-round uLBC-128 is about  $2^{113.1}$ , and  $2^{128.6}$  for 9-round. Therefore, it is expected that the algorithm does not have an effective differential trail of 9 rounds, and 4 rounds of the algorithm can spread to all active boxes. Therefore, the differential (or linear) attack potentially can analyze 16 rounds of uLBC-128 at most.

#### 4.1.2 Key Recovery

The 8-round distinguisher shown in Table 22 leads to 24 active S-boxes at one round before, and 28 active S-boxes at the round after, such that, only 1-round extension can be appended to before and after the differential/linear distinguisher to launch a valid attack for 128-bit version uLBC-128. The round of attack is bounded by 10 rounds, and although some techniques can be used to extend the attack by one or two rounds, there is still 8-round redundancy to protect the security.

And for 256-bit key version uLBC-256, two rounds can be appended before and after the 8-round distinguisher (after two-round propagation in forward/backward direction, the s-boxes are all active in the state), the expected number of attack rounds is 2 + 8 + 2 = 12.

Even in the ideal case, there is a valid 8-round differential/linear distinguisher with only one active nibble at the beginning and one active nibble in the output state, the full diffusion will occur through four rounds before and after the distinguisher, for uLBC-128, the upper bound of the number of round we possible attack is 4+8+4=16. For uLBC-256, one more round could be padded at the beginning and the end, the attack round is bounded by 5+8+5=18. And for uLBC-384-t, we possibly attack at most 6+8+6=20 rounds. There are also more than two rounds redundancy.

#### 4.1.3 Differential-Linear Cryptanalysis

Differential-linear cryptanalysis [LH94] is a cryptanalytic method that combines aspects of differential cryptanalysis and linear cryptanalysis to analyze block ciphers. The method divides the cipher E into two parts,  $E = E_1 \circ E_0$ , where there exists a strong truncated differential for the first part  $E_0$  with probability  $p = Pr[\alpha \xrightarrow{E_0} \beta]$  and there exists a strongly biased linear approximation with bias  $\epsilon$  for the second part  $E_1$ , such that  $Pr[\Gamma_0 \xrightarrow{E_1} \Gamma_1] = 1/2 + \epsilon$ . The bias of differential-linear distinguisher is  $\epsilon_{\alpha,\Gamma_1} = Pr[\Gamma_1 \cdot (E(x \oplus \alpha) \oplus E(x)) = 0] - 1/2 = 4p\epsilon^2$ , where  $Pr[\Gamma_0 \cdot \beta = 0] = 1$ . The differential-linear distinguisher is effective, when  $\epsilon_{\alpha,\Gamma_1} > 2^{-n/2}$  for *n*-bit block size. Based on the number of active S-boxes for differential and linear analysis as discussed in Section 4.1.1, the differential-linear distinguisher can apply to a maximum of 8 rounds. Since the full diffusion will occur through four rounds, for uLBC-128, the upper bound of the number of rounds we may attack is 4 + 8 + 4 = 16 at most.

#### 4.2 Boomerang-type Attacks

The boomerang-type attacks treats a block cipher E as the composition of two subciphers  $E_0$  and  $E_1$ , where have two differentials  $\alpha \xrightarrow{E_0} \beta$  and  $\gamma \xrightarrow{E_1} \delta$  with probabilities pand q respectively. Then the probability of a boomerang distinguisher is estimated by  $Pr[E^{-1}(E(x) \oplus \delta) \oplus E^{-1}(E(x \oplus \alpha) \oplus \delta) = \alpha] = p^2q^2$ . The probability of  $p = Pr[\alpha \xrightarrow{E_0} \beta]$ and  $q = Pr[\gamma \xrightarrow{E_1} \delta]$  is mainly bound by the active S-boxes in the differential. According to Table 17, we estimate the longest boomerang-type distinguisher will not exceed 8-rounds (3 + 1 + 4). In fact, we test the boomerang probability with the sandwich attack [DKS10] which introduces a sub-cipher  $E_m$  in the middle, utilizing our 4-round differentials with the minimal number of active S-boxes, and found there is no boomerang distinguisher with (4+1+4)-form can return in a valid probability. Since the full diffusion will occur through four rounds, for uLBC-128, the upper bound of the number of rounds we possibly attack is 4+8+4=16 at most.

### 4.3 Impossible Differential Attack

Impossible differential attack [BBS99] finds two internal state differences  $\Delta_f$ ,  $\Delta_r$  such that  $\Delta_f$  will never propagated to  $\Delta_r$ .

The full diffusion will occur through four rounds of uLBC, so we can construct 8-round impossible differential trails. In Table 18, we show an 8-round impossible differential. The zero-correlation trail of the cipher is the same as the impossible differential trail. It can construct 8 rounds of zero-correlation trails. Therefore, 16 rounds are analyzed at most for uLBC-128.

Position	State
$\Delta_f$	*00000000000000000000000000000000000000
Round 1	0***00000000000000000000000000000000000
Round 2	0000*0**00000000000000**0****00000
Round 3	*0**0******0??**0***0000**0***??
Round 4	??????*?????*??????????????????
Round 4	****?00***0*0*?00**?0?******0*0
Round 5	000*0**00*00*000000*00000*0*000*
Round 6	00000000000000*00*000000*000000
Round $7$	*00000000000000000000000000000000000000
$\Delta_r$	0aaa0000000000000000000000000000000000

Table 18: An impossible differential trail of uLBC.

## 4.4 Meet-in-the-Middle Attack

Partial-matching [AS08] cannot work if the number of rounds reaches full diffusion rounds in each of the forward and backward directions. The 4-round full diffusion property with our low-latency S-boxes and diffusion layer enables us to claim that any inserted key bit of K non-linearly affects all bits of the state after 4-round propagation in the forward or the backward directions for uLBC.

Thus, partial-matching can work for at most (4-1) + (4-1) + 1 = 7 rounds. The condition for the initial structure (IS) [SA09], also called independent biclique [BKR11], is that key differential trails in the forward direction and those in the backward direction do not share active non-linear components. For uLBC-128, since any key differential affects all 32 S-boxes after at least 4 rounds in the forward and backward directions, there is no such differential that shares active S-boxes in more than 4 rounds. Thus, the number of rounds used for IS is bounded by 4. Assuming that the splice-and-cut technique allows an attacker to add 4 extra rounds in the worst case, at most 15-round (4 + 4 + 7) MitM attack may be feasible. Because a whitening key is added at the end of the encryption, we consider it is difficult to launch a 15-round attack on uLBC-128.

#### 4.5 Integral Attack

Integral attack [DKR97] prepares a set of plaintexts so that particular cells can contain all the values in the set and the other cells are fixed to a constant value. We search for the optimal integral distinguisher based on the division property [XZBL16, HWW20]. Modelling the division property by Mixed-Integer-Linear-Programming (MILP) we get a 10-round integral distinguisher for uLBC-128 with 127 active bits at input:

$$C^{1}|A^{31}|A^{32}|A^{32}|A^{32},$$
  

$$\downarrow 10r$$
  

$$B^{4}|U^{28}|U^{32}|U^{32}|U^{32},$$

where A denotes all values in the cell appear exactly the same number, B denotes the sum of all values in the multiset is 0, C denotes the value of the bit is fixed through the multiset and U denotes no particular property exists. With this 10-round integral distinguisher, we can launch an attack with 4 rounds of key-filter at the end and one round adding to beginning with statistical method. Therefore, we can attack at most 15 rounds of  $\mu$ LBC-128.

We focus on differential-like and linear-like cryptanalysis and give differential cryptanalysis, boomerang attack, linear cryptanalysis, impossible differential cryptanalysis, integral cryptanalysis, and related-key cryptanalysis of uLBC algorithm and results are in Table 19. The round function of the uLBC-256 is similar to uLBC-128, but the difference is the key schedule. Therefore, the uLBC-256 focuses on the impact of key length change in the key recovery process and related key attacks. For all versions of uLBC, sufficient security redundancy is reserved.

Crymtanalyzia methoda	Round of	Estimated	Estimated analysis round (Upper bound)				
Cryptanarysis methods	distinguisher	uLBC-128	uLBC-256	uLBC-384t			
Differential attack	8	16	18	20			
Linear attack	8	16	18	20			
Differential-linear attack	8	16	18	20			
Boomerang attack	8	16	18	20			
Impossible differential attack	8	16	18	20			
Zero correlation attack	8	16	18	20			
Meet-in-the-middle attack	7	15	17	19			
Integral attack	10	14	16	17			

Table 19: Summary of analysis results of uLBC

#### 4.6 Related-key (Related-tweak) cryptanalysis of uLBC-384t

Under the related-key scenario, we introduce an MILP model to search for the minimized number of active S-boxes, the lower bound of active S-boxes for each round is shown in Table 20. For related-tweak scenarios, the attacker has the weaker ability, and the number of rounds of attack will not exceed that under the related-key scenario. With the results, we are able to prove strong bounds against related-tweak linear and differential attacks. In particular, no related tweak linear or differential distinguisher based on a characteristic is possible for 14-round uLBC-384t, which already exceeds 64 active S-boxes.

And, for the related-tweak boomerang distinguisher or differential-linear distinguisher, we can construct at most a 6 + 1 + 6 = 13-round distinguisher, which has 6-round upper part and 6-round lower part, and 1-round middle part without intersection of S-boxes. The probability is at least  $2^{-104}$ , the 14-round distinguisher also has probability lower than  $2^{-128}$ .

According to the length of the related-key distinguisher, at most 6 + 13 + 6 = 25 round attack can be launched when the length of the key is 384-bit.

Table 20: Estimation of the number of active S-boxes for uLBC-384t under related-tweak

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# of active S-boxes	0	0	1	4	8	13	23	32	35	39	44	51	60	>64

## 5 Implementation

Since our target construction is a low-latency block cipher, thus we want to minimize the latency of our block cipher, uLBC-128 is fully unrolled (that optimizes the signal delay from the input to output ports) and has its latency measured as a combinatorial circuit. For comparison, the latency of other ciphers with no less than 128 bits of security and block size are also included, namely QARMA<sub>9</sub>-128, QARMA<sub>11</sub>-128, Midori-128, SPEEDY-6, and AES-128. The S-box of QARMA<sub>9</sub>-128 and QARMA<sub>11</sub>-128 is the best low-latency  $\sigma_0$  in Table 16. As we can see in Table 21, uLBC-128 has the best area performance among all the ciphers and the minimum delay among the 128-bit block ciphers. We also give the decryption uLBC-DEC-128. Compared to uLBC-128, uLBC-DEC-128 has a slightly higher delay and a much larger area in the latency optimized implementation. In this section, we discuss the result of uLBC-128, and the implementation result of uLBC family in full unrolled mode, as shown Table 23 in Appendix C. Notably, the latency of uLBC-256 has lower latency than QARMA11-128. We also compared the round functions of uLBC-384t and SKINNY-128-384 in Table 24. The throughput of uLBC-384t is about twice that of SKINNY-128-384. Except for UMC55nm, the latency of uLBC-384t is lower than that of SKINNY-128-384 and  $AT^1$  performs better.

The delay and area of the ciphers are measured with Synopsys Design Compiler S-2021.06-SP3 and 2 open process libraries (Nangate 15nm and Nangate 45nm) and 2 commercial process libraries (TSMC 28nm and UMC 55nm). We follow the evaluation framework outlined in [BIL+21, ABD+23], using the set\_max\_delay/set\_max\_area instructions to reduce the delay/area. Thus, we give two results, one is the area optimized implementation and the other is the latency optimized implementation.

It can be seen that for both libraries,  $\mathsf{uLBC-128}$  has the second shortest latency (the fastest is SPEEDY-6), and the smallest area. However, since SPEEDY-7-192 is fully broken by Boura *et al.* [BDBN23], we believe its advantage in performance should be treated with caution. Compared with Midori-128,  $\mathsf{uLBC-128}$  has a delay of  $11\%\sim15\%$  shorter and  $\mathsf{uLBC-128}$  has excellent AT performance, the best performance except SPEEDY-6, and even the best performance in UMC55nm.

## 6 Conclusion

In this paper, we propose a new low-latency 128-bit block cipher, called uLBC-128, which offers a reduced implementation cost. Our experimental results consistently demonstrate that uLBC-128 exhibits lower latency and requires less area compared to  $QARMA_9-128$ , Midori128, and AES-128. Furthermore, uLBC-128 presents a higher throughput-to-area ratio when compared to these three ciphers. While the SPEEDY-6 cipher offers the lowest latency among the tested ciphers, its security arguments fall short, particularly when considering that Boura *et al.* have successfully broken the full round of SPEEDY-7-192. This raises concerns about the security of the 7-round SPEEDY. Thus, it is crucial to

<sup>&</sup>lt;sup>1</sup>AT denotes area-time product, which we set to  $AT = area(GE) \cdot Delay(ns)/10000$ .

			Area C	Optimized	1	Latency Optimized			
Platform	Cipher	Aı	rea	Delay	<u>۸</u>	A	ea	Delay	۸T
		$\mu m^2$	GE	ns	AI	$\mu m^2$	GE	ns	AI
	AES	24333	123764	1.292	$16.0 \times 10^{4}$	32637	166000	0.785	$13.0 \times 10^4$
	Midori128	5114	26011	0.889	$2.3 \times 10^{4}$	6839	34783	0.627	$2.2 \times 10^{4}$
NG	QARMA <sub>9</sub> -128	6237	31723	0.950	$3.0 \times 10^{4}$	9082	46193	0.650	$3.0 \times 10^{4}$
NanGate 15nm	QARMA <sub>11</sub> -128	7503	38162	1.170	$4.5 \times 10^{4}$	10710	54475	0.780	$4.2 \times 10^{4}$
101111	SPEEDY-6	5666	28819	0.589	$1.7 \times 10^{4}$	7576	38532	0.370	$1.4 \times 10^{4}$
	uLBC-128	4349	22120	0.740	$1.6 \times 10^{4}$	5742	29204	0.551	$1.6 \times 10^{4}$
	uLBC-DEC-128	5086	25869	0.833	$2.2{ imes}10^4$	7170	36468	0.592	$2.2 \times 10^{4}$
	AES	55702	110520	3.13	$34.6 \times 10^4$	103337	205033	1.57	$32.2 \times 10^4$
	Midori128	12936	25667	2.43	$6.2 \times 10^{4}$	34522	68496	1.22	$8.4 \times 10^{4}$
TOMO	QARMA <sub>9</sub> -128	15358	30472	2.37	$7.2 \times 10^{4}$	42191	83713	1.29	$10.8 \times 10^{4}$
28nm	QARMA <sub>11</sub> -128	18304	36318	2.83	$10.3 \times 10^{4}$	48495	96220	1.57	$15.1 \times 10^{4}$
201111	SPEEDY-6	13938	27655	1.33	$3.7 \times 10^{4}$	26616	52809	0.71	$3.7 \times 10^{4}$
	uLBC-128	10933	21693	1.98	$4.3 \times 10^{4}$	24897	49399	1.09	$5.4 \times 10^{4}$
	uLBC-DEC-128	13052	25897	2.19	$5.7 \times 10^{4}$	44003	87310	1.14	$10.0 \times 10^4$
	AES	91710	114925	14.38	$165.3 \times 10^{4}$	120052	150441	8.12	$122.2 \times 10^4$
	Midori128	18907	23693	9.06	$21.5 \times 10^{4}$	24646	30885	6.73	$20.8 \times 10^{4}$
NanCata	QARMA <sub>9</sub> -128	22929	28733	10.37	$29.8 \times 10^{4}$	33803	42360	6.87	$29.1 \times 10^{4}$
45nm	QARMA <sub>11</sub> -128	27510	34474	12.54	$43.2 \times 10^{4}$	40297	50497	8.27	$41.8 \times 10^{4}$
	SPEEDY-6	21079	26415	5.89	$15.6 \times 10^{4}$	27441	34387	3.98	$13.7 \times 10^{4}$
	uLBC-128	15955	19994	7.63	$15.3 \times 10^{4}$	19922	24964	5.74	$14.3 \times 10^{4}$
	uLBC-DEC-128	18550	23246	8.47	$19.7 \times 10^{4}$	25579	32054	6.32	$20.3 \times 10^4$
	AES	123037	109854	14.21	$156.1 \times 10^4$	294367	262828	3.74	$98.3 \times 10^4$
	Midori128	26645	23790	7.24	$17.2 \times 10^{4}$	111906	99916	2.83	$28.3 \times 10^{4}$
IMC	QARMA <sub>9</sub> -128	29173	26047	7.45	$28.2 \times 10^4$	114481	102215	2.59	$38.0 \times 10^4$
55nm	QARMA <sub>11</sub> -128	34831	31099	9.06	$28.2 \times 10^4$	137188	122489	3.10	$26.5 \times 10^4$
001111	SPEEDY-6	28859	25767	6.68	$13.2 \times 10^{4}$	108594	96959	1.56	$15.1 \times 10^{4}$
	uLBC-128	22545	20130	5.60	$11.3 \times 10^{4}$	78362	69966	2.23	$15.6 \times 10^{4}$
	uLBC-DEC-128	25251	31643	7.34	$23.2 \times 10^4$	112811	141369	2.34	$33.1 \times 10^{4}$

Table 21: Area and latency optimized cipher performance. AT denotes area-time product, which we set to  $AT = area(GE) \cdot Delay(ns)$ .

provide evidence of the security of our newly designed cipher. The uLBC block cipher offers robust security arguments against differential and linear attacks.

In terms of future directions, it would be interesting to extend our design by applying the new methods for low-latency 4-bit S-boxes to larger S-boxes, such as those with 5-8 bits. Additionally, exploring the efficient implementation of the cipher using instruction set architecture such as SIMD and AVX2 for software performance and side-channel security would also be valuable areas to investigate.

## Acknowledgements

The authors sincerely thank the anonymous reviewers of CIC 2024 for providing valuable comments to help us improve the overall quality of the paper.

This work is supported by the National Natural Science Foundation of China (Nos. 62072270 and 62302250), and the Young Elite Scientists Sponsorship Program by CAST (2023QNRC001).

## References

[ABD<sup>+</sup>23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The qarmav2 family of tweakable block ciphers. *IACR Trans. Symmetric Cryptol.*, 2023(3):25–73, 2023. URL: https://doi.org/10.46586/tosc.v2023.i3.25-73, doi: 10.46586/TOSC.V2023.I3.25-73.

- [AS08] Kazumaro Aoki and Yu Sasaki. Preimage attacks on one-block md4, 63-step MD5 and more. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers, volume 5381 of Lecture Notes in Computer Science, pages 103–119. Springer, 2008. doi:10.1007/978-3-642-04159-4\\_7.
- [Ava17] Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017. doi:10.13154/tosc. v2017.i1.4–44.
- [BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In Tetsu Iwata and Jung Hee Cheon, editors, Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II, volume 9453 of Lecture Notes in Computer Science, pages 411–436. Springer, 2015. doi:10.1007/978-3-662-48800-3\\_17.
- [BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding, volume 1592 of Lecture Notes in Computer Science, pages 12–23. Springer, 1999. doi:10.1007/3-540-48910-X\\_2.
- [BCG<sup>+</sup>12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE A low-latency block cipher for pervasive computing applications extended abstract. In Xiaoyun Wang and Kazue Sako, editors, Advances in Cryptology ASIACRYPT 2012 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings, volume 7658 of Lecture Notes in Computer Science, pages 208–225. Springer, 2012. doi:10.1007/978-3-642-34961-4 \\_14.
- [BCLR17] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. Proving resistance against invariant attacks: How to choose the round constants. In Jonathan Katz and Hovav Shacham, editors, Advances in Cryptology -CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II, volume 10402 of Lecture Notes in Computer Science, pages 647–678. Springer, 2017. doi: 10.1007/978-3-319-63715-0\\_22.
- [BDBN23] Christina Boura, Nicolas David, Rachelle Heim Boissier, and María Naya-Plasencia. Better steady than speedy: Full break of SPEEDY-7-192. In Carmit Hazay and Martijn Stam, editors, Advances in Cryptology EUROCRYPT 2023 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV, volume 14007 of Lecture Notes in Computer Science, pages 36–66. Springer, 2023. doi:10.1007/978-3-031-30634-1\\_2.

- [BDD<sup>+</sup>23] Yanis Belkheyar, Joan Daemen, Christoph Dobraunig, Santosh Ghosh, and Shahram Rasoolzadeh. Bipbip: A low-latency tweakable block cipher with small dimensions. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):326– 368, 2023. doi:10.46586/tches.v2023.i1.326-368.
- [BEK<sup>+</sup>21] Dušan Božilov, Maria Eichlseder, Miroslav Knezevic, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. Princev2 more security for (almost) no overhead. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, Selected Areas in Cryptography, Lecture Notes in Computer Science, pages 483–511. Springer, 2021. 27th International Conference on Selected Areas in Cryptography : SAC 2020, SAC 2020 ; Conference date: 19-10-2020 Through 23-10-2020. doi:10.1007/978-3-030-81652-0\_19.
- [BGLS19] Zhenzhen Bao, Jian Guo, San Ling, and Yu Sasaki. Peigen a platform for evaluation, implementation, and generation of s-boxes. *IACR Transactions on* Symmetric Cryptology, 2019(1):330–394, Mar. 2019. URL: https://tosc.i acr.org/index.php/ToSC/article/view/7406, doi:10.13154/tosc.v201 9.i1.330–394.
- [BIL<sup>+</sup>21] Subhadeep Banik, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, and Kosei Sakamoto. Orthros: A Low-Latency PRF. IACR Transactions on Symmetric Cryptology, 2021(1):37–77, Mar. 2021. URL: https://tosc.iacr. org/index.php/ToSC/article/view/8833, doi:10.46586/tosc.v2021.i1 .37-77.
- [BJK<sup>+</sup>16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, Advances in Cryptology -CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II, volume 9815 of Lecture Notes in Computer Science, pages 123–153. Springer, 2016. doi: 10.1007/978-3-662-53008-5\\_5.
- [BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings, volume 7073 of Lecture Notes in Computer Science, pages 344–371. Springer, 2011. doi:10.1007/978-3-6 42-25385-0\\_19.
- [DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher square. In Eli Biham, editor, Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings, volume 1267 of Lecture Notes in Computer Science, pages 149–165. Springer, 1997. doi:10.1007/BFb0052343.
- [DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3g telephony. In Tal Rabin, editor, Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings, volume 6223 of Lecture Notes in Computer Science, pages 393–410. Springer, 2010. doi:10.1007/978-3-642-14623-7\\_21.

- [GJN<sup>+</sup>15] Jian Guo, Jérémy Jean, Ivica Nikolic, Kexin Qiao, Yu Sasaki, and Siang Meng Sim. Invariant subspace attack against full midori64. IACR Cryptol. ePrint Arch., page 1189, 2015. URL: http://eprint.iacr.org/2015/1189.
- [GL16] David Gérault and Pascal Lafourcade. Related-key cryptanalysis of midori. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings, volume 10095 of Lecture Notes in Computer Science, pages 287–304, 2016. doi:10.1007/978-3-319 -49890-4\\_16.
- [HWW20] Kai Hu, Qingju Wang, and Meiqin Wang. Finding bit-based division property for ciphers with complex linear layers. IACR Trans. Symmetric Cryptol., 2020(1):396-424, 2020. doi:10.13154/tosc.v2020.i1.396-424.
- [JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II, volume 8874 of Lecture Notes in Computer Science, pages 274–288. Springer, 2014. doi:10.1007/978-3-662-45608-8\\_15.
- [LH94] Susan K. Langford and Martin E. Hellman. Differential-linear cryptanalysis. In Yvo Desmedt, editor, Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings, volume 839 of Lecture Notes in Computer Science, pages 17–25. Springer, 1994. doi:10.1007/3-540-48658-5\\_3.
- [LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):510–545, 2021. doi:10.46586/tches.v2021.i 4.510-545.
- [LS16] Meicheng Liu and Siang Meng Sim. Lightweight MDS generalized circulant matrices. In Thomas Peyrin, editor, Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers, volume 9783 of Lecture Notes in Computer Science, pages 101–120. Springer, 2016. doi:10.1007/978-3-662-52993-5\\_6.
- [QDW<sup>+</sup>22] Lingyue Qin, Xiaoyang Dong, Anyu Wang, Jialiang Hua, and Xiaoyun Wang. Mind the TWEAKEY schedule: Cryptanalysis on skinnye-64-256. In Shweta Agrawal and Dongdai Lin, editors, Advances in Cryptology - ASIACRYPT 2022
  28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I, volume 13791 of Lecture Notes in Computer Science, pages 287–317. Springer, 2022. doi:10.1007/978-3-031-22963-3\\_10.
- [Ras22] Shahram Rasoolzadeh. Low-latency boolean functions and bijective s-boxes. IACR Trans. Symmetric Cryptol., 2022(3):403-447, 2022. URL: https://doi. org/10.46586/tosc.v2022.i3.403-447, doi:10.46586/T0SC.V2022.I3.4 03-447.
- [SA09] Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In Antoine Joux, editor, Advances in Cryptology - EU-ROCRYPT 2009, 28th Annual International Conference on the Theory and

Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings, volume 5479 of Lecture Notes in Computer Science, pages 134–152. Springer, 2009. doi:10.1007/978-3-642-01001-9\\_8.

- [SSH99] Ivan Sutherland, Bob Sproull, and David Harris. Logical Effort: Designing Fast CMOS Circuits. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.
- [TAY17] Mohamed Tolba, Ahmed Abdelkhalek, and Amr M. Youssef. Improved multiple impossible differential cryptanalysis of midori128. IEICE Trans. Fundam. Electron. Commun. Comput. Sci., 100-A(8):1733-1737, 2017. URL: https: //doi.org/10.1587/transfun.E100.A.1733, doi:10.1587/TRANSFUN.E10 0.A.1733.
- [XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I, volume 10031 of Lecture Notes in Computer Science, pages 648–678, 2016. doi: 10.1007/978-3-662-53887-6\\_24.

## A Test Vectors

## A.1 uLBC-128

	Plaintext	: 0000000	00000000	00000000	00000000
1.	Key	: 00000000	00000000	0000000	00000000
	Ciphertext	: e6b6b249	c25994a4	25c480b8	52f4899a
	Plaintext	: 2d33f9e1	4c935dc3	150e07ef	14909c24
2.	Key	: 2f5a39f6	5ef0d47b	4c2ae6b5	21fb0a08
	Ciphertext	: 5384bf47	150a1f05	059c1e70	4894ca21

#### A.2 uLBC-256

	Plaintext	: 00000000	00000000	00000000	00000000
1	Key	: 00000000	00000000	00000000	00000000
1.		00000000	00000000	00000000	00000000
	Ciphertext	: fb0be725	0cfd59a9	d2712fb1	899ff396
	Plaintext	: 2d643085	0cad830b	53764dad	7be5c3a9
2	Key	: 2990cbcb	613391be	00a582cd	722e610f
2.		42e65a59	2135ab5f	4a85fb01	222224e0
	Ciphertext	: b6cfa6a5	90dba5ea	a5a7004e	5a8339f4

## A.3 uLBC-384t

1.	Plaintext Key	: 00000000 : 00000000 00000000 00000000	00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000	00000000 00000000 00000000 00000000
	Ciphertext	: 64adeb02	32294d9c	1e268365	764e674a
	Plaintext	: 2d7d941a	0f864e4b	3a7e25e3	2fe97815
	Key	: 08cac346	37da2d2c	54cbc059	0cc13e8d
2.		62f1a185	65a35582	4642fdef	582d79f2
		11a12c18	6c55e4b2	7b1999ae	4db8ada5
	Ciphertext	: 6bfe00ba	63397825	379e397e	fa11d15a

# B Differential trails with minimize number of active sboxes for uLBC-128

Table 22 shows the differential trails with the minimum number of active S-boxes for  $\mathsf{uLBC-128}$  with 3 to 7 Rounds.

26

# of Round	Position	State
	$\Delta_{in}$	00000000c0020000000000000000000
3r	Round 1	000000000000000000000000000000000000000
	Round 2	222000000000000000000000000000000000000
	$\Delta_{out}$	0666101100000000000550500000000
	$\Delta_{in}$	0002000000000000200000000002000
	Round 1	000000000000000000000000000000000000000
4r	Round 2	0000000000660600000000000000000
	Round 3	ccc000000000999000020220000000
	$\Delta_{out}$	3ffc6066110111100555563500006606
	$\Delta_{in}$	00040010120020000100004200000400
	Round 1	000000000040010000000040000000
5~	Round 2	000000000000000000000000000000000000000
57	Round 3	00000002022000000000000000000000
	Round 4	0000000088800001101000000001110
	$\Delta_{out}$	0000aa82404480882202088881992220
	$\Delta_{in}$	0000000000000000800000100108000
	Round 1	00000000000011000000022000000
	Round 2	080800000000000000000000000000000000000
6r	Round 3	000008800000000000000000020020000
	Round 4	0000000000000002022111022020888
	Round 5	01100088088000002002800800001100
	$\Delta_{out}$	00009198323102028080891926460000
	$\Delta_{in}$	0000000000000000100000200102000
	Round 1	00000000000011000000011000000
	Round 2	010100000000000000000000000000000000000
7r	Round 3	0000022000000000000000000020020000
• •	Round 4	00000000000000010111110110101111
	Round 5	01100088011000001001400400008800
	Round 6	00000101020204044040808020200000
	$\Delta_{out}$	00880000008801108800440000001001
	$\Delta_{in}$	10000018000000002000800200000
	Round 1	0001000002000000000000000000000000
	Round 2	000000000008088000000011100000
	Round 3	080800000002020000000000002313
8r	Round 4	00000220101101111110000080081101
	Round 5	001100001100000100008000010200
	Round 6	0000000000810110000010044400000
	Round 7	080800000008080000000000004c00
	$\Delta_{out}$	000080886066022200000008cc42202

Table 22: Differential trails with the minimized number of active S-boxes.

# C Implementation of **uLBC** Family

Table 23 shows the implementation of  $\mathsf{uLBC}$  family. Table 24 shows the implementation of round function of  $\mathsf{uLBC}\textsc{-384t}$  and SKINNY-128-384.

			Area (	Optimize	d	Latency Optimized			
Platform	Cipher	Aı	rea	Delay	۸T	Aı	rea	Delay	٨T
		$\mu m^2$	GE	$\mathbf{ns}$	AI	$\mu m^2$	GE	ns	AI
NanCata	uLBC-128	4349	22120	0.740	$1.6 \times 10^4$	5742	29204	0.551	$1.6 \times 10^4$
15nm	uLBC-256	6146	31262	0.964	$3.0 \times 10^{4}$	8052	40953	0.681	$2.8 \times 10^{4}$
101111	uLBC-384t	8863	45078	1.302	$5.9 \times 10^4$	11527	58628	0.927	$5.4 \times 10^{4}$
TOMO	uLBC-128	10933	21693	1.98	$4.3 \times 10^{4}$	24897	49399	1.09	$5.4 \times 10^4$
28nm	uLBC-256	15913	31574	2.32	$7.3 \times 10^{4}$	36150	71726	1.34	$9.6 \times 10^{4}$
201111	uLBC-384t	22961	45558	3.16	$14.4 \times 10^{4}$	47189	93628	1.84	$17.2 \times 10^{4}$
NanCata	uLBC-128	15955	19994	7.63	$15.3 \times 10^{4}$	19922	24964	5.74	$14.3 \times 10^{4}$
45nm	uLBC-256	22704	28451	9.83	$28.0 \times 10^{4}$	28165	35295	7.15	$25.2 \times 10^4$
101111	uLBC-384t	32833	41144	13.3	$54.7 \times 10^{4}$	39983	50104	9.62	$48.2 \times 10^{4}$
IMC	uLBC-128	22545	20130	5.60	$11.3 \times 10^{4}$	78362	69966	2.23	$15.6 \times 10^{4}$
55nm	uLBC-256	31609	28223	7.30	$20.6 \times 10^4$	105104	93843	2.77	$26.0 \times 10^{4}$
	uLBC-384t	45611	40724	10.17	$41.4 \times 10^{4}$	146099	130446	3.79	$49.4 \times 10^{4}$

Table 23: Area and Latency Optimized Cipher Performance of uLBC family. AT denotes area-time product, which we set to  $AT = area(GE) \cdot Delay(ns)$ .

Table 24: Area and Latency Optimized Cipher Performance of the round function of uLBC-384t and SKINNY-128-384. AT denotes area-time product, which we set to  $AT = area(GE) \cdot Delay(ns) \cdot (Rounds + 1)$ .

			Latency Optimized						
Platform	Cipher	Rounds	Ar	rea	Delay	Throughput	۸T		
			$\mu m^2$	GE	$\mathbf{ns}$	Mb/s	AI		
NanGate	uLBC-384t	30	1923	9780	0.057	68866	$1.7{ imes}10^4$		
$15 \mathrm{nm}$	SKINNY-128-384	56	1297	6598	0.064	33296	$2.4 \times 10^{4}$		
TSMC	uLBC-384t	30	5446	10806	0.11	35798	$3.7 \times 10^4$		
28 nm	SKINNY-128-384	56	5790	11489	0.12	17847	$7.9 \times 10^{4}$		
NanGate	uLBC-384t	30	7266	9105	0.55	7160	$15.6 \times 10^4$		
45 nm	SKINNY-128-384	56	4583	5743	0.71	3016	$23.2 \times 10^4$		
UMC	uLBC-384t	30	14357	12819	0.31	12702	$12.3 \times 10^4$		
55nm	SKINNY-128-384	56	8219	7339	0.28	7649	$11.7{\times}10^4$		