The New Low Latency Block Cipher uLBC

Anonymous Submission

Abstract. How do we design a secure encryption algorithm whose hardware implementation is fast with less cost? We apply the low latency and lightweight design strategy to design a block cipher with high throughput in hardware, and the block cipher can be implemented with less area. We apply 4-bit S-boxes to optimal the depth of the circuit, and propose a new method to constructing low-latency coordinate functions. By means of proper combination of coordinate functions, a large number of low-latency S-boxes are constructed. For the difference diffusion of round function with nibble permutation and 4×4 binary matrices with branch number of 4, we study the choice of the nibble permutation, and propose a well-choose method for the nibble permutation which can guarantee the impossible differential and differential are 8 round at most. Finally, we introduce a new 128-bit block cipher family **uLBC** with 128-bit and 256-bit key, which has 4 versions, where **uLBC-128s** has smaller latency and area than both QARMA$_9$-128, Midori-128 and AES-128, and has a higher throughput-area ratio than other ciphers under Nangate 45nm and Nangate 15nm.

Keywords: block cipher design · low latency · low latency S-box · lightweight

1 Introduction

Block cipher is a fundamental cryptographic primitive, which is widely used to provide the confidential running as software or hardware. Block cipher design is complex, and we should have a well thought-out strategy to balance many factors such as cryptographic strength, latency, implementation cost, power, and energy consumption, etc. Which trade-offs are the right ones to make is determined by the application.

For some applications with real-time requirements, some low latency ciphers are required, that can instantaneously encrypt a given plaintext, i.e., the entire encryption and decryption should take place within the shortest possible delay. low latency ciphers have drawn increasing attention in recent years. Besides, these ciphers take great advantages in some high-throughput cases such as the Internet of Things (IoT) and encrypted hardware components of CPUs. In 2012, Borghoff et al. proposed the first low latency block cipher called Prince [BCG+12], which propose the α-reflection structure, i.e. the encryption and decryption are similar with the keys K and $K \oplus \alpha$, respectively. The round function of prince basically follows the AES structure, with the exception of using a 4×4 almost MDS binary matrices with branch number 4 instead of MixColumns mapping. After that, a series of ciphers are put forward, such as Midori [BBI+15], MANTIS [BJK+16], QARMA [Ava17] and SPEEDY [LMMR21a] etc..

The block cipher Midori [BBI+15] proposed in ASIACRYPT 2015, is a family of two block ciphers with different block length: Midori64 and Midori128. Midori128 applies AES-like design, with the 8-bit S-boxes as the confusion module and 4×4 almost MDS binary matrices as the diffusion module. The 8-bit S-boxes are consisting 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 threat to the security of Midori. Guo et al. gave the invariant subspace attack against the full block cipher Midori64 [GJN+15]. Gérault et al. gave practial attack on Midori64 and Midori128 under related key model [GL16]. MANTIS with 64-bit block length, proposed in CRYPTO 2016 [BJK+16], is a low latency variant of the SKINNY family. MANTIS is a tweakable block cipher for memory encryption, which is enhanced with a TWEAKEY framework [JNP14] for Prince, and apply the involutory MIDORI S-box to optimize for small area and low circuit depth. QARMA [Ava17] is inspired by Prince and MANTIS, but uses a three-round Even-Mansour scheme instead of an FX-construction. In SAC 2020, the authors of Prince proposed Princev2 [BEK+21] and improved the security of Prince without changing the number of rounds or rotation operations. Recently, some new low latency block ciphers are proposed. SPEEDY [LMMR21b] has no requirements for area and decryption speed. It incorporates CMOS hardware into ciphers design and applies a high-speed 6-bit S-box which is implemented using NAND gates. Although SPEEDY has lower delay for encryption, Boura et al. give a differential attack for the full round SPEEDY-7-192, which break the expected security strength [BDBNP22]. For some specially application, Belkheyar et al. [BDD+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. Besides the block ciphers, Banik et al. introduced a low latency pseudorandom function Orthros [BIL+21a], which input a 128-bit message and a 128-bit key and output a 128-bit random number. Orthros applies two 128-bit permutation to construct the pseudorandom function. For low lantency, the permutation has 12 rounds, which takes 4-bit S-boxes, bit permutation, nibble permutation, and 4×4 almost MDS binary matrices.

This paper is focused on low latency block ciphers with 128 bit block length, which can achieve more high throughput with a small area in hardware. 4-bit S-boxes are normally used in lightweight ciphers. To our best knowledge, there are mainly two methods to select a 4-bit low latency S-box. Thanks to the small size, the security properties of 4-bit S-boxes have been widely studied. Among all 4-bit S-boxes, some are optimal against differential and linear attacks with high algebra degree and can be divided into 16 equivalence classes [Saa12], making it convenient to search an S-box that satisfies the security level as well as special requirements. Hence the first method is choosing several equivalence classes and then selecting low latency S-boxes from these optimal S-boxes. The second method is constructing a series of low latency S-boxes and then selecting one with the best security properties. When constructing low latency S-boxes, designers can use different metrics to measure latency of coordinate functions. The most popular metric is depth, which is used to estimate the path delay of the S-box based on the number of transistors to be performed sequentially in the operation. However, it treats NAND and NOR equally, which needs to be improved, just as analyzed in [LMMR21b].

Our contribution. This paper proposes a new 128-bit block ciphers, which has the lowest latency among the 128-bit block ciphers as far as we know.

How do we design a secure encryption algorithm whose hardware implementation is fast with less cost? We apply the low latency and lightweight design strategy to design a block cipher with high throughout in hardware, and the block cipher can be implementation with less area.

We apply 4-bit S-boxes to optimal the depth of the circuit. A new method to constructing low latency coordinate functions is proposed. By means of proper combination of coordinate functions, a large number of low latency S-boxes are constructed. Then we perform cryptographic analysis on those S-boxes and select one with the best security characteristics. We hope that our results will inspire others to design new and efficient low latency cryptographic primitives.

For 128-bit block, there are 32 nibbles. We design the diffusion module by nibble permutation and the lightweight 4×4 almost MDS binary matrices. The 4×4 almost MDS binary matrices has been used in the design of Prince, MIDORI, MANTIS, Orthros etc. The branch number of this transform is 4, thus, an active nibbles can propagate 27 nibbles at most after three rounds. Hence, we want an active nibble can diffuse to any one of all 32 nibbles (This case is usually called full diffusion), the round number is 4 at least. In this case, the selecting of a nibble permutation resistant to the differential attack is of great importance. We propose a well-choose method for the nibble permutation which can guarantee both the impossible differential and differential are 8 rounds at most. Hence, the differential path and impossible differential path are both has the same length.

Besides, we give an optimal key schedule against related key attack and weak key attack with low implementation cost. The key schedule is only consisting of a nibble permutation for 128-bit key, and it's easy to extend to 256-bit key version with XOR operations and a LFSR function for nibbles. The LFSR function has been used in SKINNY [BJK+16].

We give fully-unrolled hardware implementation for ciphers MIDORI, QARMA, and AES etc. Among them our cipher is the lowest latency among the 128-bit block ciphers[1] with the same security claim, and the throughout is the highest seen Table 9.

2 Descriptions of Block Cipher

The **uLBC** is a Substitution Permutation Network (SPN) based block cipher. SPN structure is based on principles of confusion and diffusion implemented in use of substitution and linear transformation, respectively. In this section, we first give an overview, and then introduce encryption and decryption algorithms and the key schedule algorithms respectively.

2.1 Overview

The block cipher **uLBC** is a 128-bit block with a 128-bit key and a 256-bit key. According to the different key lengths, we denote them as **uLBC-128** and **uLBC-256**. For the security of the block ciphers, especially for low latency application, it is not relevant for the related, known and chosen-key security, that is called NRK security. The low latency ciphers MIDORI, QARMA, Prince and SPEEDY all state NRK security. Hence, we can reduce the rounds of the cipher to obtain the high performance for NRK security. Otherwise, we call it strong security. The relevant parameters are shown in Table 1.

Table 1: The parameters of **uLBC**.

Scheme	Block size	Key size	Round	Security
uLBC-128	128 bits	128 bits	24	Strong
uLBC-128s	128 bits	128 bits	20	NRK security
uLBC-256	128 bits	256 bits	30	Strong
uLBC-256s	128 bits	256 bits	24	NRK security

The computation of **uLBC** is big-endian. The generation rules of bit string to word are as follows: every eight consecutive bit strings $b_0b_1b_2b_3b_4b_5b_6b_7$ form a byte, where the most significant bit is b_0 and the least significant bit is b_7. The conversion from byte to word is big-endian, a format in which the most significant byte is stored first in memory.

The 128-bit block can be represented as 32 nibbles, which is illustrated by the matrix in Figure 1.

2.2 Encryption

The **uLBC** block cipher is based on the SPN structure. First, the plaintext block P and the white key RK_0 are exclusive or, and then N rounds (round functions) are iterated. The **round function** includes S-box substitution (SubNib), nibble permutation (PosPerm), column mixing (MixColumn), and round key addition operations (RKAdd). The round

[1]Our implementations are publicly available in `https://anonymous.4open.science/r/CLB-8057`.

$$\begin{pmatrix} s_0 & s_4 & s_8 & s_{12} & s_{16} & s_{20} & s_{24} & s_{28} \\ s_1 & s_5 & s_9 & s_{13} & s_{17} & s_{21} & s_{25} & s_{29} \\ s_2 & s_6 & s_{10} & s_{14} & s_{18} & s_{22} & s_{26} & s_{30} \\ s_3 & s_7 & s_{11} & s_{15} & s_{19} & s_{23} & s_{27} & s_{31} \end{pmatrix}$$

Figure 1: **uLBC** state.

function is illustrated in Figure 2 and the encryption is given in Algorithm 1. The subkeys $RK_1, RK_2, \ldots, RK_N$ are round keys generated by the key schedue algorithm. The final output is the ciphertext C. We provide test vectors of **uLBC** in Appendix A.

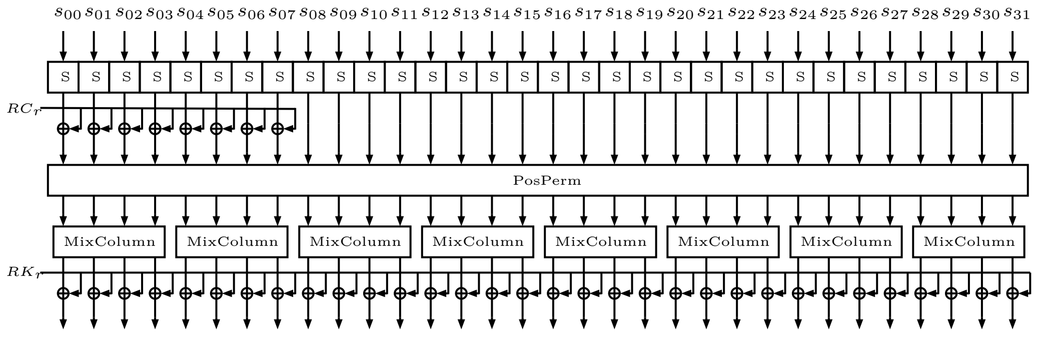

s_{00} s_{01} s_{02} s_{03} s_{04} s_{05} s_{06} s_{07} s_{08} s_{09} s_{10} s_{11} s_{12} s_{13} s_{14} s_{15} s_{16} s_{17} s_{18} s_{19} s_{20} s_{21} s_{22} s_{23} s_{24} s_{25} s_{26} s_{27} s_{28} s_{29} s_{30} s_{31}
S
RC_r
PosPerm
MixColumn
RK_r

Figure 2: The round function of **uLBC**.

Algorithm 1 The Encryption of **uLBC**.

```
Input: plaintext: $P$, round key: $RK_0, \ldots, RK_N$
Output: ciphertext: $C$
  $X_0 = P \oplus RK_0$
  for $i = 0$ to $N - 1$ do
    // this is the $r$-th round ($r = i + 1$)
    $Y_i = \text{SubNib}(X_i)$
    $Z_i = \text{AddConst}(Y_i)$
    $U_i = \text{PosPerm}(Z_i)$
    $W_i = \text{MixColumn}(U_i)$
    $X_{i+1} = W_i \oplus RK_{i+1}$ // round key addition
  end for
  $C = X_N$
```

SubNib. Substitution referred to as S-box is the only nonlinear operation based on nibbles, which divides 128-bit state X into 32 nibbles $X = \{x_0, x_1, \ldots, x_{31}\}$. Use the nibbles to query the S-box, and obtain the output state Y, where $Y = \text{SubNib}(X) = \{S(x_0), S(x_1), \ldots, S(x_{31})\}$. The S-box substitution table is in Table 2.

Table 2: S-box substitution table.

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output	11	8	10	0	15	14	2	1	9	12	13	4	3	6	5	7

AddConst. The first two columns are exclusive or 32-bit constant $c = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$ which is constructed based on 6-bit LFSR, and the feedback polynomial is $(a_0, a_1, a_2, a_3, a_4, a_5) \rightarrow (a_5 \oplus a_4 \oplus 1, a_0, a_1, a_2, a_3, a_4)$. The round constants corresponding to (c_1, c_2) are shown in Table 3.

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

Round	0	1	2	3	4	5	6	7
(c_1, c_2)	0x80	0xc0	0xe0	0xf0	0xf8	0x7c	0xbc	0xdc
Round	8	9	10	11	12	13	14	15
(c_1, c_2)	0xec	0xf4	0x78	0x3c	0x9c	0xcc	0xe4	0x70
Round	16	17	18	19	20	21	22	23
(c_1, c_2)	0xb8	0x5c	0xac	0xd4	0x68	0x34	0x18	0xc0
Round	24	25	26	27	28	29	30	31
(c_1, c_2)	0x84	0x40	0xa0	0xd0	0xe8	0x74	0x38	0x1c

For **uLBC-128**, $(c_2, c_3, c_4, c_5) = $ 0x5a5a, for **uLBC-128s**, $(c_2, c_3, c_4, c_5) = $ 0xc5c5, for **uLBC-256**, $(c_2, c_3, c_4, c_5) = $ 0xa3a3, **uLBC-256s**, $(c_2, c_3, c_4, c_5) = $ 0x3c3c. It is used to distinguish the algorithm versions corresponding to different versions.

The round constant corresponding to (c_6, c_7) consists of the decimal part of π, as shown in the Table 4.

Table 4: The last two bytes of round constant addition.

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

$Z = \text{AddConst}(Y)$, where $z_i = y_i \oplus c_i$, $i = 0, 1, 2, 3$, and $z_i = y_i$, $i = 4, 5, \ldots, 31$.

PosPerm. After **AddConst**, the state goes to nibble permutation. The position of nibble permutation is in Table 5.

Table 5: The position of byte permutation.

Input x									Output $p(x)$							
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

$U = \text{PosPerm}(Z) = \{u_0, u_1, \ldots, u_{31}\}$, where $u_i = z_{p(i)}$, $i = 0, 1, \ldots, 31$.

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

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix},$$

it's obvious that the matrix $M^{-1} = M$.

$W = \{w_0, w_1, \ldots, w_{31}\} = \text{MixColumn}(U)$, that is:

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

If the nibbles 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 none zero nibbles among the input and output nibbles. Hence, the branch number is 4.

RKAdd. During the encryption process of the r-th round ($r = 1, 2, ..., N$), the state W after column mixing operation and the round key RK_r are bitwise exclusive or to obtain the output state.

2.3 Decryption

The decryption algorithm is the inverse operation of the encryption algorithm. Perform N rounds (round functions) of iterative operations, including decryption round key addition, the inversion of column mixing inversion (MixColumn^{-1}), the inversion of nibble permutation (PosPerm^{-1}), and the inversion of S-box (SubNib^{-1}). Finally, output the exclusive or with the key RK_0 to obtain the plaintext. The pseudo code of the decryption algorithm is in Algorithm 3.

Algorithm 2 The Decryption of **uLBC**.

```
Input: ciphertext: $C$, round key: $RK_0, \ldots, RK_N$
Output: plaintext: $P$
  $X_N = C$
  for $i = N - 1$ to 0 do
    // this is the $r$-th round ($r = N - i$)
    $W_i = X_{i+1} \oplus RK_{i+1}$ // round key addition
    $U_i = \text{MixColumn}^{-1}(W_i)$
    $Z_i = \text{PosPerm}^{-1}(U_i)$
    $Y_i = \text{AddConst}(Z_i)$
    $X_i = \text{SubNib}^{-1}(Y_i)$
  end for
  $P = X_0 \oplus RK_0$
```

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

MixColumn^{-1} Same as MixColumn, because of $M^{-1} = M$, multiply each column of the intermediate state matrix with the 4×4 binary matrix M.

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

Table 6: The position of byte permutation inversion.

Input x									Output $p(x)$							
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

Algorithm 3 The Decryption of **uLBC**.

```
Input: ciphertext: $C$, round key: $RK_0, \ldots, RK_N$
Output: plaintext: $P$
  $X_N = C$
  for $i = N - 1$ to 0 do
    // this is the $r$-th round ($r = N - i$)
    $W_i = X_{i+1} \oplus RK_{i+1}$ // round key addition
    $U_i = \text{MixColumn}^{-1}(W_i)$
    $Z_i = \text{PosPerm}^{-1}(U_i)$
    $Y_i = \text{AddConst}(Z_i)$
    $X_i = \text{SubNib}^{-1}(Y_i)$
  end for
  $P = X_0 \oplus RK_0$
```

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 exclusive or, and the constant value is in the opposite order to the encryption.

SubNib^{-1} The S-box substitution inversion calculation is a non-linear substitution based on nibbles. The 32 nibbles query the S-box inversion one by one, and obtain the output state $X = \text{SubNib}^{-1}(Y)$.

2.4 Key Schedule

In order to reduce the area and improve the efficiency, the key schedule algorithm adopts nibble permutation. The key schedule of **uLBC-256** is slightly more complex than that of **uLBC-128**. We use the f function for nibble, which is based on LFSR construction with a cycle of 15.

For **uLBC-128**, the key length is equal to the block length. Its key schedule algorithm uses function F and key K to generate round key RK_i, $i = 0, 1, ..., N$. The details are below:

$$RK_0 = K,\ RK_i = F(RK_{i-1}),\ i = 1, \ldots, N.$$

The F function is a nibble permutation. The 128 bit state is divided into 32 nibbles. The nibble based matrix has four rows and eight columns. The nibble position permutation 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 pseudo code of round function $Y = F(X)$ is $Y[i] = X[\text{PermK}[i]]$, $i = 0, 1, \ldots, 31$.

For **uLBC-256**, the key length is twice the block length. The key schedule algorithm first divides the $2n$ length master key K into two n bit round keys $K = K_0||K_1$, and then uses F function and f function to generate the round key. The f function is the transformation of nibble, and the F function is the same as **uLBC-128**. The details are below:

$$RK_0 = K_0 \oplus K_1,\ RK_i = F^i(K_0 \oplus f^i(K_1)),\ i = 1, \ldots, N - 1.$$

Table 7: Nibble permutation table PermK.

Input x									Output PermK[i]							
0	4	8	12	16	20	24	28		18	19	25	24	31	30	11	10
1	5	9	13	17	21	25	29	$\rightarrow$	0	1	22	23	3	2	6	7
2	6	10	14	18	22	26	30		4	5	12	13	26	27	20	21
3	7	11	15	19	23	27	31		15	14	9	8	17	16	29	28

f is constructed based on 4-bit LFSR, feedback polynomial $(a_0, a_1, a_2, a_3) \rightarrow (a_0 \oplus a3, a0, a1, a2)$. The corresponding inputs and outputs are in Table 8.

Table 8: Nibble substitution table for f.

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output	0	8	1	9	2	10	3	11	12	4	13	5	14	6	15	7

3 Design Decisions

3.1 Main Ideas and Strategies of Design

According to the known block ciphers, It is obviously that the implementation of the SPN construction is more efficient than that of the Feistel construction, which is adopted by the international cryptographic algorithm standards AES [DR98], PRESENT [KL11], etc. Besides, most low latency ciphers based on the SPN sturcture with some improvement to support encryption and decryption using the same implementation. For SPN structure, the security and the implementation both depend on the round function. We apply 4-bit S-boxes to optimal the depth of the circuit. In order to provide stronger bounds on the number of active S-boxes, we select nibble permutation carefully and use 4×4 almost MDS binary matrices as the diffusion function. Although several design choices have been borrowed from existing ciphers, we give some new methods to optimized for low latency and lightweight. For the 4-bit S-boxes, a new method to constructing low latency coordinate functions is proposed, which is more consistent with the implementation. By means of proper combination of coordinate functions, a large number of low latency S-boxes are constructed. For the diffusion function, the choice of nibble permutation has work to the bounds on the number of active S-boxes. We propose a well-choose method for the nibble permutation which can guarantee the impossible differential and differential are both 8 rounds at most.

The algorithm supports 128-bit block, and the key supports 128 bits and 256 bits. We apply a very simple key schedule by nibble permutation for 128-bit key, which generates the key diffusion that is effectively resistant to related key attacks and weak key attacks. For 256-bit key, we make use of the nibble permutation to extend the Tweakey frame of SKINNY[BJK$^+$16]. Besides, a round-specific constant chosen according to the methods suggested by Beierle et al. [BCLR17] is added to resist invariant subspace or nonlinear invariant attacks.

Besides, we make a tradeoff among the security, hardware implementation and software implementation. The design scheme is simple, convenient for hardware and software implementation, and flexible. Under the condition of ensuring the security of the block cipher, the algorithm has good performance on ASIC hardware platforms and software platform with AVX instructions.

3.2 The Design of S-boxes

The 4-bit S-box has good cryptographic properties, occupies a small area, has low latency, and the cost of increasing side-channel protection is relatively low, which is used in the design of many cryptographic algorithms. The **uLBC** algorithm also uses a 4-bit S-box. The latency of the S-box is very low, and security features are considered, including differential probability, linear bias advantage, algebraic number, fixed point, etc. The security indicators of S-box are as follows:

1. The maximum difference probability is 2^{-2}.
2. The maximum linear approximation probability 2^{-2}.
3. The highest algebraic degree of S-box is 3, and the highest algebraic degree of S-box inverse is 3.
4. The S-box is a permutation without a fixed point.

3.2.1 Logic Gates and Their Latency

In CHES 2021, Leander et al. [LMMR21b] analyze the latency of logic gates by calculating Fan-in-to-Latency Ratio (FLR) for each gates. They point out the shortcoming of depth metric used in [BBI+15] and observe that NAND and OAI gates get the highest FLR scores among all the logic cells tested. Finally, only NAND are considered in the construction of their 6-bit S-boxes.

When building 4-bit S-boxes, we found that NAND does have the lowest latency but there are some logic gates that also perform well in term of latency, which are ignored in Leander et al.' paper. Traditional depth metric is based on the number of the transistors to be sequentially proceeded in the operation and only considers gates with fan-in 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.

Method of Logical Effort, proposed by Sutherland et al.[SSH99], ia a method for designing MOS circuits to achieve high speeds. This method simplifies the MOS circuit as networks of resistance and capacitance for circuit analysis. The delay of a logic gate d consists of two parts, a fixed part called the *parasitic delay* p and the other part called the *effort delay* f. The total delay is the sum of the effort and parasitic delays, defined as

$$d = f + p.$$

Then the *logical effort* g and *electrical effort* h are introduced. The logical effort describes the computational cost due to the circuit topology, while the electrical effort describes the cost of driving capacitive loads. The effort delay is the product of these two factors:

$$f = gh.$$

The logical effort of the inverter is defined as 1 delay unit so that all logical efforts are measured relative to the delay of a simple inverter. Since the logical effort of a gate can be computed by how much more input capacitance a gate presents to deliver the same output current as an inverter, we get logical efforts of other gates based on the ratio of input capacitances. Table 9 gives assumed values of logical effort of some gates. Electrical effort is also called fanout by many CMOS designers. Note that for simplification, we assume that it is fixed to 1 in the following.

The parasitic delay of a logic gate is fixed, and the main contribution is the capacitance of the source/drain regions of the transistors that drive the gates output. The parasitic delay is a multiple of the parasitic delay of the inverter which is set to 1 delay unit. Table 9 gives assumed values of parasitic delay of some gates.

Table 9: Values of delay factors for some logical gates

Gate	Fanin	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
AOI22	4	2.0	4.0

For logic networks, the path logical effort is the product of the logical effort of all the logic gates along the path. We use uppercase symbol G to represent the path logical effort. Thus we have

$$G = \prod g_i. \tag{1}$$

where the subscripts index the logic gates along the path. Further, the path effort is defined and represented by the uppercase symbol F. F is computed by the equation:

$$F = GBH,$$

where $BH = \prod h_i$. Similarly, the path parasitic delay P is defined as

$$P = \sum p_i. \tag{2}$$

For an N-stage logic network, the minimum path delay can be computed by

$$D = NF^{\frac{1}{N}} + P. \tag{3}$$

3.2.2 S-box Searching

Using the path delay, the first step is to find a number of coordinate functions for 4-bit S-boxes with low latency. We select some common logic gates, as shown in Table 9, and construct Boolean functions based on the delays of all the logic gates along the path of the Boolean function. Table 9 lists values of delay factors for all the logic gates we selected. According to (3), the delay of a Boolean function D is derived as

$$D = NF^{\frac{1}{N}} + P.$$

In our search, due to the inconvenience of path delays, we iterate based on the depths of logic gates. We assume that the depths of INV, NAND2/NOR2, NAND3/NOR3/OAI21/AOI21 and NAND4/NOR4/OAI22/AOI22 are weighted as 0.5, 1, 2 and 2. The initial functions

$$y = x_0, \quad y = x_1, \quad y = x_2, \quad y = x_3,$$

and the INV functions

$$y = \neg x_0, \quad y = \neg x_1, \quad y = \neg x_2, \quad y = \neg x_3,$$

are set to have factors:

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

Table 10: Properties of some low latency S-boxes

S-box	Depth	D	Latency	
			15nm	45nm
Orthros	3.5	10	4.95496	0.09573
InvOrthros	4	13	9.04691	0.14699
MidoriSb0	3.5	10	5.26489	0.10603
MidoriSb1	4	12	7.68204	0.1066
QARMAσ_0	3.5	11	4.95496	0.11188
QARMAσ_1	4	12	7.27624	0.1259
QARMAσ_2	4.5	13	7.8453	0.14516
InvQARMAσ_2	4	13	9.04691	0.14699
S_0	3.5	10	4.77033	0.09286
Inv S_0	3.5	11	5.46202	0.11876
S_1	3.5	10	4.77033	0.09427
Inv S_1	3.5	12	6.19784	0.11249

Then we search Boolean functions starting with depth 1 until 4.5. 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 funtions 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 output is 0-1 balanced.
3. The function expression contains all 4 bits.
4. The algebraic degree is greater than 1.

After that, we use these Boolean functions to construct 4-bit S-boxes. Note that in order to reduce the search space, we require that the coordinate functions of the S-box are sorted by truthtable during the search. Our selected S-boxes are listed below:

S_0	0,8,9,13,2,10,14,12,4,11,1,15,7,3,6,5,
S_1	11,8,10,0,15,14,2,1,9,12,13,4,3,6,5,7,

The S-box S_0 meets the indicators 1-3 while S_1 meets all 4 indicators of S-box. Considering security and other characteristics, we choose S_1 as the S-box of the **uLBC** algorithm.

Table 10 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 10 are further divided into two categories $D = 10$ and $D = 11$, and the S-boxes with depth 4 are further divided into categories $D = 12$ and $D = 13$. We observed that the latency of the S-boxes are basically in line with our estimates. Generally, S-boxes with high D values take an advantage in latency, while there are a few exceptions, such as MidoriSb1, which has lower latency than those S-boxes with $D = 11$. Besides, in the experiments, QARMAσ_2 has a lower latency than QARMA2Inv, but the depth value of QARMAσ_2 is greater than that of InvQARMAσ_2 and did not match the measured values. When using our metric, both S-boxes have the same D values, which is at least consistent with the experimental results of 45nm.

As shown in Table 10, the S-box S_0 is superior to the other S-boxes on the list, and its inverse S-box has the same D value as QARMAσ_0, although the actual measurement results are slightly worse than QARMAσ_0. Then we replace the S-box in the QARMA round function with S_0 and measure its latency, as listed in Table 11. The measurement of the round function further verifies the D values of S-boxes.

Table 11: Latency of the QARAMA round funtion with different S-boxes

S-box	Latency			
	Encryption		Decryption	
	15nm	45nm	15nm	45nm
QARMAσ_0	33.72543	0.37704	20.81541	0.29382
S_0	32.48224	0.36264	21.98062	0.29137

3.3 The Design of Diffusion Layer

For the low latency and lightweight design strategy, We design the diffusion module by 4×4 almost MDS binary matrices, which has been used in the design of Prince, MIDORI, MANTIS, Orthros etc. However the 4×4 almost MDS binary matrices has branch number of 4 less than the MDS code, which result in a 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+21b]. 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-boxes number in the differential or linear path. Here, we applied the nibble permutation. There are $32! = 2^{117}$ permutaion for 32 nibbles. It is impossible to traverse. Banik et al. give some analysis to select good nibble permutation, such as Contion 1.

Condition 1 (Condition 3[BIL+21b]). *For each column $(u_{4i}, u_{4i+1}, u_{4i+2}, u_{4i+3})$ in the 4×8 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.

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

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

Condition 3. *For each active S-box input 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,

$$1 \to 3 \to 9 \to (18 \sim 27) \to 32.$$

Hence, the 4 round diffusion are need to be taken into account. We give the following condition.

Condition 4. *For each active S-box input in the first round, it will propagate the difference to all nibbles 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 make sure the full diffusion, which is used to limit the length of an impossible differential.

Because **MixColumn** can mix the column, we choose the row permutation independently for different rows satisfying all conditions in the above. There are about $8! = 40320$ row permutations. We search all this cases, and there are about 695520 permutations in consistent with all 4 conditions. Then we search the differential path by estimate the active S-boxes for 6,7,8,9 rounds. We found when the number of active S-boxes is 40 for 6-round **uLBC** with a selected nibble permutation, It's hard to find 9-round differential for with high probability. The permutation found is row-independent, therefore, we can implement the round function with the AVX instructions efficiently.

Table 12 shows the 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 4 rounds and 5 rounds. Our nibble permutation cause a stronger diffusion at the same time, which lead to a 8-round impossible differential. However we find many 9-round impossible differential for Branch1/2 of Orthros, respectively, seen Table 13 and Table 14, where '*' means the none zero difference nibble, '0' means the zero difference nibble, '?' means the nibble difference is unkonw, and $a \neq 0$.

Table 12: Comparison of lower bounds of the number of active S-boxes.

	number of active S-box						rounds of IMP
rounds	4	5	6	7	8	9	
Midori128 [BBI+15]	16	20	30	35	38	-	7 [TAY17]
Orthros Branch1 [BDD+23]	16	25	36	50	60	-	9 Table 13
Orthros Branch2 [BDD+23]	16	25	36	51	60	-	9 Table 14
Our nibble permutation	16	25	40	52	60	66	8 Table 16

Table 13: An impossible differential trail of Branch1.

Position	State
Δ_f	0000000000000000000000000*000000
Round1	000000000000*0**0000000000000000
Round2	00000000***000000***00000000**0*
Round3	0***0000*0*****0?**?*0**?**?***0
Round4	????????***0?????????????*?*????
Round4	????*??????*???????*?????*???*??
Round5	?*0***0?****0**000?*0*?***00****
Round6	0*0000*000*00**0*0000*0*000*0000
Round7	00*0*0000*0000000000000000000000
Round8	0000000000000000000000000*000000
Δ_r	000000000000a0aa0000000000000000

Table 14: An impossible differential trail of Branch2.

Position	State
Δ_f	00000000000000*00000000000000000
Round*	00000000000000000***000000000000
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*000000*0000**0000*0000**00
Round7	0000000*000*0*000000000000000000
Round8	*0000000000000000000000000000000
Δ_r	000000000000000000000000aa0a0000

3.4 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. Although a little complexity key schedule does not increase encryption latency, we apply a simple key schedule by nibble permutation for 128-bit key to reduce the cost of implementation.

For 256-bit key, we make use of the nibble permutation to extend the Tweakey frame of SKINNY[JNP14] with tweakey 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 S-box subtitute f is used in each nibble of the second parts for each round. f is constructed by an 4-bit LFSR with a period of 15. Hence, the mastkey can be computed by any two consecutive round subkeys.

For the nibble permutation in uLBC-128, uLBC-256, we choose the same nibble permutation to reduce implementation cost. Hence, we add some conditions on the nibble permutation in key schedule to remove some iteration path in the related-key setting to sieve good nibble permutation. Then we check the nibble permutation by the number of active sboxes in the related-key setting.

Many lightweight block ciphers apply a very simple key schedule in which the round keys only differ by addition of a round-specific constant, which is vulnerable to invariant subspace attack. Beierle et al.[BCLR17] analyzes the resistance of some ciphers against invariant attacks and give a method to find round constants which guarantee the resistance to all types of invariant attacks. Following it, 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 peiod of 63 to resist the invariant attacks; the last part is used as random numbers.

4 Security Analysis

This section give cryptanalysis of the block cipher **uLBC**, including differential cryptanalysis [BS91], boomerang attack [Wag99], linear cryptanalysis [Mat94], impossible differential cryptanalysis [Knu98, BBS99a], integral cryptanalysis [DKR97, KW02] and related-key cryptanalysis [Bih94].

Differential/Linear cryptanalysis. In order 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 15. Because of the equivalence between differential and linear trails, the result for the linear attack is the same.

Table 15: 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	66

We prove that the minimum number active S-boxes in 9-round **uLBC-128** is at least 66, and the maximum differential probability is 2^{-132}. 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.

Impossible differential cryptanalysis. Impossible differential attack [BBS99b] finds two internal state differences Δ_f, Δ_r such that Δ_f will never propagated to Δ_r.

The full diffusion will occur through four rounds of **uLBC**, so we can construct 8-round impossible differential trails. In Table 16, 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.

Table 16: An impossible differential trail of **uLBC**.

Position	State
Δ_f	*0000000000000000000000000000000
Round1	0***0000000000000000000000000000
Round2	0000*0**000000000000**0****00000
Round3	*0**0******0??**0***0000**0***??
Round4	??????*?????*???????????????*?*?
Round4	****?00***0*0*?00**?0?*******0*0
Round5	000*0**00*00*0000000*00000*0*00*
Round6	000000000000000*00*000000*000000
Round7	*0000000000000000000000000000000
Δ_r	0aaa0000000000000000000000000000

Integral cryptanalysis. 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-Programmin (MILP) and 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 key-filter at the end and one round adding to beginning with statistical method. Therefore, we can attack at most 15 rounds of **uLBC-128**.

Related-key cryptanalysis. In the case of related keys, MILP modeling is carried out and differential active boxes of related keys are searched. The number of boxes is in Table 17, where '?' means that it is not the smallest number of active S-boxes. Therefore, **uLBC-128**'s related-key boomerang attack can construct at most $4+5+1 = 10$ rounds of distinguisher, and the complexity of 11 rounds exceeds 2^{-128}. For **uLBC-256**'s related-key boomerang attack, a maximum of $5 + 6 + 1 = 12$ rounds of distinguisher can be constructed, and the complexity of 13 rounds exceeds 2^{-128}.

Table 17: Estimation of the number of active boxes under related-key

Round	1	2	3	4	5	6	7	8	9	10	11
Number of **uLBC-128**	1	4	6	11	19	28	37?	50?	-	-	-
Number of **uLBC-256**	0	1	3	6	11	19	28	37?	50?	-	-

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 18. The round function of the **uLBC-256** is similar to **uLBC-128**, but the difference is 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.

Table 18: Summary of analysis results of **uLBC**

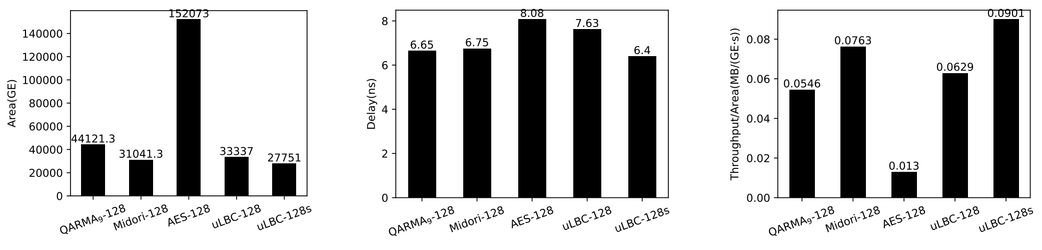
Attack	Round of distinguisher	Estimated analysis round	
		uLBC-128	**uLBC-256**
Differential cryptanalysis	8	16	18
Boomerang attack	8	16	18
Linear cryptanalysis	8	16	18
Impossible differential cryptanalysis	8	16	18
Integral cryptanalysis	10	14	16
Related-key cryptanalysis	10/12	18	20

5 Implementation

Since we want to minimize the latency of our block cipher, **uLBC-128**(**uLBC-128s**) is fully unrolled 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$_9$-128, Midori-128 and AES-128.

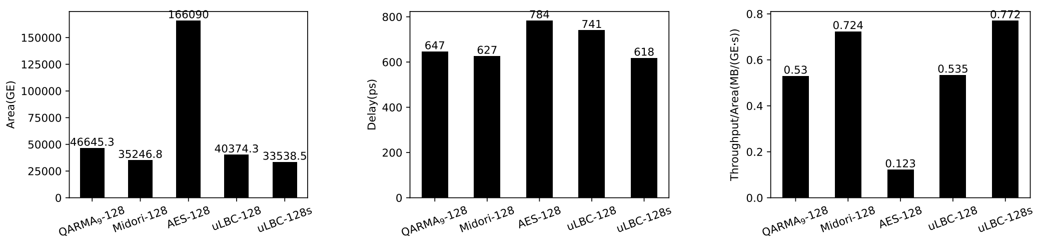
The delay and area of the ciphers are measured with *Synopsys Design Compiler S-2021.06-SP3* and 2 process libraries(Nangate 45nm and Nangate 15nm). First we model a fully unrolled cipher as a combinatorial circuit, create virtual clocks with very short period and zero input&output delays, then use the commands `uniquify` and `compile_ultra` to optimize it. The delay and area of **uLBC-128**(**uLBC-128s**) and other ciphers are displayed in figure 3 and figure 4.

It can be seen that for both Nangate 45nm and Nangate 15nm, **uLBC-128s** has smaller latency and area than both QARMA$_9$-128 and Midori-128, and has a higher throughput-area ratio than other ciphers. Compared with low latency Midori-128, the delay is reduced by 5% and the throughput-area ratio is increased by 18%.



(a) Area (b) Delay (c) Throughput-Area Ratio

Figure 3: Performance of **uLBC** and other ciphers measured with Nangate 45nm



(a) Area (b) Delay (c) Throughput-Area Ratio

Figure 4: Performance of **uLBC** and other ciphers measured with Nangate 15nm

6 Conclusion

In this paper, we propose a new low latency 128-bit block cipher with less implementation cost. **uLBC-128s** almost always has smaller latency and area than both QARMA$_9$-128, Midori-128 and AES-128 in our experiment, and has a higher throughput-area ratio than the other 3 ciphers. Here, we propose a new method to constructing low latency coordinate functions, which is more consistent with the experiment than the depth metric used in [BBI+15]. We apply it to construct the 4-bit S-box and optimal the depth of the circuit, and give a large number of low latency S-boxes. Then we perform cryptographic analysis on those S-boxes and select one with the best security characteristics. Besides, for the diffusion module by nibble permutation and 4×4 almost MDS binary matrices with branch number 4, We propose a well-choose method for the nibble permutation which can guarantee the impossible differential and differential are 8 round at most. It is interesting the differential path and the impossible differential path are both the same length.

We hope that our results will inspire others to design new and efficient low latency cryptographic primitives.

References

[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.

[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.

[BBS99a] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 12–23. Springer, 1999.

[BBS99b] 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.

[BCG+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 Yalçın. Prince – a low-latency block cipher for pervasive computing applications. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, pages 208–225, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[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*, pages 647–678, Cham, 2017. Springer International Publishing.

[BDBNP22] Christina Boura, Nicolas David, Rachelle Heim Boissier, and Maria Naya-Plasencia. Better steady than speedy: Full break of speedy-7-192. Cryptology ePrint Archive, Paper 2022/1351, 2022. https://eprint.iacr.org/2022/1351.

[BDD+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.

[BEK+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.

[Bih94] E. Biham. New types of cryptanalytic attacks using related keys. *Journal of Cryptology*, 1994.

[BIL+21a] Subhadeep Banik, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, and Kosei Sakamoto. Orthros: A low-latency prf. *IACR Transactions on Symmetric Cryptology*, 2021(1):3777, Mar. 2021.

[BIL+21b] 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, 2021.

[BJK+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.

[BS91] E. Biham and A. Shamir. Shamir, a.: Differential cryptanalysis of des-like cryptosystems. journal of cryptology 4(1), 3-72. *Journal of Cryptology*, 4(1):3–72, 1991.

[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.

[DR98] Joan Daemen and Vincent Rijmen. Rijndael, aes proposal, 1998. http://www.nist.gov/aes.

[GJN+15] Jian Guo, Jérémy Jean, Ivica Nikoli, Kexin Qiao, Yu Sasaki, and Siang Meng Sim. Invariant subspace attack against full midori64. Cryptology ePrint Archive, Paper 2015/1189, 2015. https://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*, pages 287–304, Cham, 2016. Springer International Publishing.

[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.

[JNP14] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Tweaks and keys for block ciphers: The tweakey framework. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 274–288, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[KL11] Lars R. Knudsen and Gregor Leander. *PRESENT – Block Cipher*, pages 953–955. Springer US, Boston, MA, 2011.

[Knu98] Lars Knudsen. Deal-a 128-bit block cipher. *complexity*, 258(2):216, 1998.

[KW02] Lars Knudsen and David Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption*, pages 112–127, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[LMMR21a] 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 Cryptol. ePrint Arch.*, 2021:960, 2021.

[LMMR21b] 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.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 386–397, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[Saa12] Markku-Juhani O. Saarinen. Cryptographic analysis of all 4×4-bit s-boxes. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography*, pages 118–133, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[SSH99] I. Sutherland, Bob Sproull, and David Harris. Logical effort: Designing fast cmos circuits. 1999.

[TAY17] Mohamed F. Tolba, Ahmed Abdelkhalek, and Amr M. Youssef. Improved multiple impossible differential cryptanalysis of midori128. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 100-A:1733–1737, 2017.

[Wag99] David Wagner. The boomerang attack. In Lars Knudsen, editor, *Fast Software Encryption*, pages 156–170, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[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.

A Test Vectors

Table 19 presents test vectors for **uLBC-128**, **uLBC-128s**, **uLBC-256** and **uLBC-256s**.

Table 19: Test vectors for **uLBC** in hex.

	uLBC-128
Plaintext	`00000000000000000000000000000000`
Key	`00000000000000000000000000000000`
Ciphertext	`85a67195ff9f85d378bc4181e0bc7125`
	uLBC-128s
Plaintext	`00000000000000000000000000000000`
Key	`00000000000000000000000000000000`
Ciphertext	`fecbc67e35ee273a10760dd37eed12e4`
	uLBC-256
Plaintext	`00000000000000000000000000000000`
Key	`0000000000000000000000000000000000000000000000000000000000000000`
Ciphertext	`2e5c8766b83889cadb8b826be5e21fa0`
	uLBC-256s
Plaintext	`00000000000000000000000000000000`
Key	`0000000000000000000000000000000000000000000000000000000000000000`
Ciphertext	`b7556ad14dd6b697969a0267535af186`