Exercice de style

Pierre Karpman

Exercice de style

Pierre Karpman[⚔]

Abstract

We present the construction and implementation of an 8-bit S-box with a differential and linear branch number of 3. We show an application by designing Fly, a simple block cipher based on bitsliced evaluations of the S-box and bit rotations that targets the same platforms as Pride, and which can be seen as a variant of Present with 8-bit S-boxes. It achieves the same performance as Pride on 8-bit microcontrollers (in terms of number of instructions per round) while having 1.5 times more equivalent active S-boxes. The S-box also has an efficient implementation with SIMD instructions, a low implementation cost in hardware and it can be masked efficiently thanks to its sparing use of non-linear gates.

Keywords. Block cipher design, Lai-Massey S-box, bitsliced implementation, SPN.

1 Introduction

Since the late 1990's and the end of the AES competition, the academic community and the industry have been provided with excellent block ciphers. In most cases where a cipher is needed, AES [Nat01] can readily be used and there is currently little need for a replacement. Consequently, the symmetric cryptographic community shifted focus to *e.g.* the wider picture of *authenticated encryption* through the Caesar competition, or to more specific niche applications of block ciphers. In the latter case, a popular topic is the design of "lightweight" block ciphers intended to be implemented on low-cost, resource-constraint devices. An early successful example following this trend is the block cipher Present [BKL+07], which can be implemented in small hardware circuits. Most lightweight algorithms similarly target one (or a few) platform(s) on which they are expected to perform particularly well; good performance in other cases are however not usually expected and lightweight ciphers are in general not very versatile. Typical platforms of interest include hardware circuits and 8-bit to 32-bit microcontrollers.

In this work, we design a conceptually simple block cipher targeting efficient implementations on 8-bit microcontrollers while also being expected to achieve reasonable performance in hardware. The chief academic proposal to date for this scenario is the Pride block cipher, that was presented at CRYPTO 2014. Our block cipher is built around Littlun-1, a compact 8-bit S-box with branch number 3. This allows to define a round function similar to a scaled-up variant of Present, composing the S-box application with a simple bit permutation[1]. This offers a trade-off between hardware and light software implementations: Littlun-1 is more expensive in hardware than (two

[⚔]Inria and École Polytechnique, France & Nanyang Technological University, Singapore; pierre.karpman@inria.fr. Partially supported by the Direction Générale de l'Armement and by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

[1]As a bit permutation obviously does not increase the number of active bits, an important part of the diffusion in such a cipher is played by the S-box. The typical measure of the quality of the diffusion of an S-box is its "branch number" which plays a role similar to the minimum distance of the linear codes used in AES-like designs.

applications of) the S-box of PRESENT, but the bit permutation is simple to implement with 8-bit rotations. Owing to Golding, we name our block cipher FLY.

Excluding on-the-fly key expansion, the round function of FLY costs 4 instructions less to implement than PRIDE's on AVR. Using the good branch number of LITTLUN-1, we can show that with a similar number of rounds, FLY is more resistant than PRIDE to statistical (differential and linear) attacks. This is all the more relevant as the security margin of PRIDE seems to be quite thin [ZWWD14]. Taking the key-schedule into account, one round of FLY costs one more instruction than one round of PRIDE. However, unlike PRIDE, we do not use an FX construction for the key-schedule and thus the generic security of FLY does not decrease with the amount of data available to the adversary (Dinur also showed how the FX construction can lead to more efficient time-memory-data trade-offs [Din15]).

Related work. The block cipher literature is so numerous that most new proposal will bear some similarity with past designs. In that respect, apart from PRESENT, FLY is also quite similar to RECTANGLE [ZBL+14], which also combines a SERPENT-like bitsliced application of an S-box with a rotation-implemented bit permutation. However, the S-box in RECTANGLE is on 4 bits, it does not have a branch number of 3 and the rotations are on 16-bit words. The construction of the LITTLUN S-box uses the Lai-Massey structure from the IDEA block cipher [LM91]; this structure was already used to build the second S-box of the WHIRLPOOL hash function [BR03] and the S-box of the block cipher FOX [JV04].

Structure of this paper. We present our construction of the LITTLUN S-box family in Sec. 3 and the implementation of LITTLUN-1 in Sec. 4. Sec. 5 is devoted to the design and analysis of the block cipher FLY.

2 Preliminaries

We start by defining the main notions that will be used in evaluating the cryptographic properties of our construction. Although we will mostly consider S-boxes as defined over binary strings, we may see an n-bit S-box as a mapping $\mathbf{F}_2^n \to \mathbf{F}_2^n$ whenever convenient.

Definition 1 (Differential uniformity of an S-box) *Let $\mathcal{S}$ be an n-bit S-box. We define its* difference distribution table *(or DDT) as the function $\delta_{\mathcal{S}}$ defined extensively by:*

$$\delta_{\mathcal{S}}(a,b) := \#\{x \in \mathbf{F}_2^n | \, \mathcal{S}(x) + \mathcal{S}(x+a) = b\}.$$

The differential uniformity Δ *of $\mathcal{S}$ is defined as:*

$$\max_{(a,b)\neq(0,0)} \delta_{\mathcal{S}}(a,b).$$

Put another way, an n-bit S-box with differential uniformity Δ has a maximal differential probability of $\Delta/2^n$ over its inputs.

Definition 2 (Linearity of an S-box) *Let $\mathcal{S}$ be an n-bit S-box. We define its* linear approximation table *(or LAT) as the function $\mathcal{L}_{\mathcal{S}}$ defined extensively by:*

$$\mathcal{L}_{\mathcal{S}}(a,b) := \sum_{x \in \mathbf{F}_2^n} (-1)^{\langle b, \mathcal{S}(x)\rangle + \langle a, x\rangle}.$$

The linearity ℓ *of $\mathcal{S}$ is defined as:*

$$\max_{(a,b)\neq(0,0)} \mathcal{L}_{\mathcal{S}}(a,b).$$

Roughly speaking, the linearity measures the maximum (absolute) difference between how many times a (non-trivial) linear approximation takes the value 1 and how many times it takes the value 0. It is therefore twice the difference between 2^{n-1} (for an n-bit S-box) and how many times either value is taken. In particular, if we define the *bias* b of a probability p as $|p - 1/2|$, it means that the bias of any linear approximation of an n-bit S-box of linearity ℓ is upper-bounded by $(\ell/2)/2^n$.

Definition 3 (Differential branch number of an S-box) *The* differential branch number *of an S-box $\mathcal{S}$ is:*

$$\min_{\{(a,b)\neq(0,0)|\ \delta_{\mathcal{S}}(a,b)\neq 0\}} \mathrm{wt}(a) + \mathrm{wt}(b),$$

where $\mathrm{wt}(x)$ *is the Hamming weight of* x*.*

Definition 4 (Linear branch number of an S-box) *The* linear branch number *of an S-box $\mathcal{S}$ is:*

$$\min_{\{(a,b)\neq(0,0)|\ \mathcal{L}_{\mathcal{S}}(a,b)\neq 0\}} \mathrm{wt}(a) + \mathrm{wt}(b).$$

Definition 5 (Algebraic normal form) *Let $f \in \mathbf{F}_2^n$ be an n-bit Boolean function, its* algebraic normal form *(or ANF) is defined as the polynomial $g \in \mathbf{F}_2[x_0, x_1, \ldots x_{n-1}]/ < x_i^2 - x_i >_{i<n}$ such that for all $x \in \mathbf{F}_2^n$, $f(x) = g(x[0], \ldots, x[n-1])$. Similarly, the ANF of an n-bit S-box $\mathcal{S}$ is the sequence of the ANFs of its n constituent Boolean functions $\langle \mathcal{S}(\cdot), e_i \rangle$ (with (e_i) the canonical basis of $\mathbf{F}_2^n$).*

3 The Littlun S-box construction

We present the design rationale for our S-box and an instantiation as the "Littlun-1" S-box.

3.1 The Lai-Massey structure

Our S-box uses the *Lai-Massey structure*, which was proposed in 1991 for the design of the block cipher IDEA [LM91]. The structure is similar in its objective to a Feistel or Misty ladder as it allows to construct n-bit functions out of smaller components. It is in particular well-suited to build efficient 8-bit S-boxes from 4-bit S-boxes all the while amplifying the good cryptographic properties of the 4-bit S-boxes. It was already used as such for the design of the second S-box of the Whirlpool hash function [BR03] (an early version of Whirlpool used a randomly-generated S-box), using five 4-bit S-boxes (see Fig. 3.1a) and for the design of the S-box of the Fox block cipher [JV04] which uses a three-round iterated structure. In our construction, we use the more classical variant of the structure with only three S-boxes (see Fig. 3.1b), which allows nonetheless to square the differential probability and the linearity of the underlying 4-bit S-box[2].

The choice of the Lai-Massey structure was mainly motivated by our objective of building an S-box with a branch number of three[3]. Indeed, it is easy to see that the S-box will have this property for the differential branch number by construction as soon as the 4-bit S-boxes have branch number three, and such S-boxes are well-known (see *e.g.* Serpent [BAK98]). So much cannot be said however for the linear branch number, as no (differential and linear optimal) 4-bit S-box exists with this property[4]. In fact, we are not aware of previous examples of 8-bit S-boxes with this feature either.

[2] We do not require the 4-bit S-boxes to be orthomorphisms of any group; hence we in fact only partially adhere to the Lai-Massey structure as defined by Vaudenay [Vau99].

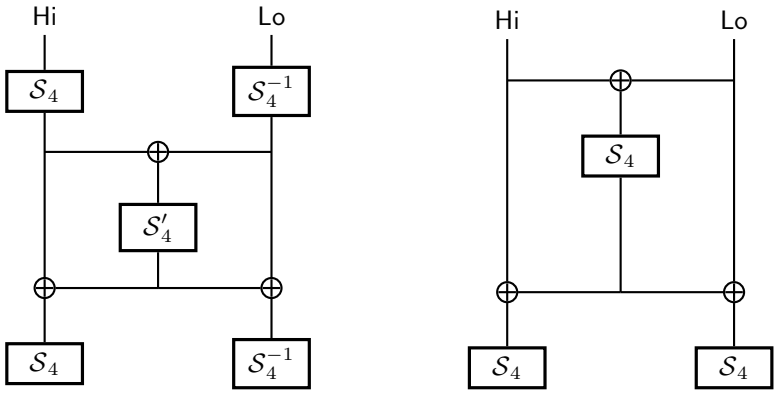
[3] This will in turn be useful to design a good lightweight round function.

[4] As demonstrated by an exhaustive search we performed on the optimal classes [LP07].

Other good properties of the structure are that it yields S-boxes with a circuit depth of two S-boxes and it allows for efficient vector implementations using SIMD instructions (see Sec. 4.3). On the downside, it requires the 4-bit S-boxes to be permutations if we want the 8-bit S-box to be one. Canteaut, Duval and Leurent recently showed how the absence of such a restriction for Feistel ladders could be used to build compact S-boxes with particularly low differential probability [CDL15]. We should note however that for the applications we have in mind (see Sec. 5), the linearity of the S-box is as important as the differential probability, and the S-box of Canteaut *et al.* is average in that respect (and in particular not better than ours). Finally, we should mention that the good and bad points of the Lai-Massey structure cited so far are shared with the Misty ladder. Choosing Lai-Massey in our case was mainly due to a matter of taste, though it is noteworthy that Misty yields S-boxes with a rather sparse algebraic expression (meaning that the polynomials in the ANF of such S-boxes tend to have many zero coefficients).

3.2 An instantiation: LITTLUN-1

We now define LITTLUN-1, a concrete instantiation of the Lai-Massey structure which achieves a differential and linear branch number of three. Although we have seen that we could guarantee this in the differential case by using a 4-bit S-box of branch number three, this is actually not necessary and we use instead a very compact member of the *class 13* of Ullrich *et al.* [UDI+11]. This S-box uses only 4 non-linear and 4 XOR gates, which is minimal for an optimal S-box of this size. This leads to an 8-bit S-box using 12 non-linear and 24 XOR gates. We give the table of the 4-bit S-box in Fig. 3.2 and of the complete 8-bit S-box in Fig. 3.3, and conclude this section by a summary of the cryptographic properties of LITTLUN-1.



(a) Lai-Massey as in the WHIRLPOOL second S-box

(b) Lai-Massey as in the LITTLUN construction

Figure 3.1: The Lai-Massey structure

```
uint8_t littlun1_s4[16] =
                {0x0, 0xa, 0x4, 0xf, 0xc, 0x7, 0x2, 0x8,
                 0xd, 0xe, 0x9, 0xb, 0x5, 0x6, 0x3, 0x1};
```

Figure 3.2: The 4-bit S-box used in LITTLUN-1 as a C array

```
uint8_t littlun1[256] =
       {0x00, 0x9b, 0xc2, 0x15, 0x5d, 0x84, 0x4c, 0xd1,
        0x67, 0x38, 0xef, 0xb0, 0x7e, 0x2b, 0xf6, 0xa3,
        0xb9, 0xaa, 0x36, 0x78, 0x2f, 0x6e, 0xe3, 0xf7,
        0x12, 0x5c, 0x9a, 0xd4, 0x89, 0xcd, 0x01, 0x45,
        0x2c, 0x63, 0x44, 0xde, 0x02, 0x96, 0x39, 0x70,
        0xba, 0xe4, 0x18, 0x57, 0xa1, 0xf5, 0x8b, 0xce,
        0x51, 0x87, 0xed, 0xff, 0xb5, 0xa8, 0xca, 0x1b,
        0xdf, 0x90, 0x6c, 0x32, 0x46, 0x03, 0x7d, 0x29,
        0xd5, 0xf2, 0x20, 0x5b, 0xcc, 0x31, 0x04, 0xbd,
        0xa6, 0x41, 0x8e, 0x79, 0xea, 0x9f, 0x68, 0x1c,
        0x48, 0xe6, 0x69, 0x8a, 0x13, 0x77, 0x9e, 0xaf,
        0xf3, 0x05, 0xcb, 0x2d, 0xb4, 0xd0, 0x37, 0x52,
        0xc4, 0x3e, 0x93, 0xac, 0x40, 0xe9, 0x22, 0x56,
        0x7b, 0x8d, 0xf1, 0x06, 0x17, 0x62, 0xbf, 0xda,
        0x1d, 0x7f, 0x07, 0xb1, 0xdb, 0xfa, 0x65, 0x88,
        0x2e, 0xc9, 0xa5, 0x43, 0x58, 0x3c, 0xe0, 0x94,
        0x76, 0x21, 0xab, 0xfd, 0x6a, 0x3f, 0xb7, 0xe2,
        0xdd, 0x4f, 0x53, 0x8c, 0xc0, 0x19, 0x95, 0x08,
        0x83, 0xc5, 0x4e, 0x09, 0x14, 0x50, 0xd8, 0x9c,
        0xf4, 0xee, 0x27, 0x61, 0x3b, 0x7a, 0xa2, 0xb6,
        0xfe, 0xa9, 0x81, 0xc6, 0xe8, 0xbc, 0x1f, 0x5a,
        0x35, 0x72, 0x99, 0x0a, 0xd3, 0x47, 0x24, 0x6d,
        0x0b, 0x4d, 0x75, 0x23, 0x97, 0xd2, 0x60, 0x34,
        0xc8, 0x16, 0xa0, 0xbb, 0xfc, 0xe1, 0x5e, 0x8f,
        0xe7, 0x98, 0x1a, 0x64, 0xae, 0x4b, 0x71, 0x85,
        0x0c, 0xb3, 0x3d, 0xcf, 0x55, 0x28, 0xd9, 0xf0,
        0xb2, 0xdc, 0x5f, 0x30, 0xf9, 0x0d, 0x26, 0xc3,
        0x91, 0xa7, 0x74, 0x1e, 0x82, 0x66, 0x4a, 0xeb,
        0x6f, 0x10, 0xb8, 0xd7, 0x86, 0x73, 0xfb, 0x0e,
        0x59, 0x2a, 0x42, 0xe5, 0x9d, 0xa4, 0x33, 0xc7,
        0x3a, 0x54, 0xec, 0x92, 0xc1, 0x25, 0xad, 0x49,
        0x80, 0x6b, 0xd6, 0xf8, 0x0f, 0xbe, 0x7c, 0x11};
```

Figure 3.3: The LITTLUN-1 S-box as a `C` array

Proposition 1 (Statistical properties) *The differential uniformity of* LITTLUN-1 *and of its inverse is 16 and its linearity is 64. Its DDT and LAT are shown in Fig. A.1 and Fig. A.2 respectively.*

In essence, Prop. 1 means that the probability (taken over all the inputs) of any non-trivial differential relation going through the S-box is upper-bounded by 2^{-4} and the bias of any non-trivial linear approximation is upper-bounded by 2^{-3}.

Proposition 2 (Diffusion properties) *The differential and linear branch number of* LITTLUN-1 *and of its inverse is 3.*

Proposition 3 (Algebraic properties) *The maximal degree of (the ANF of)* LITTLUN-1 *is 5 in four of the eight output bits, 4 in two other and 3 in the remaining two. The maximal degree of its inverse is 5 in six of the eight output bits and 4 in the other two.*

Proposition 4 (Fixed points) LITTLUN-1 *has two fixed points: 0 and 187.*

It can be seen from the DDT and LAT of LITTLUN-1 that it is quite structured in a way; it would be very unlikely to obtain an S-box with such tables if it were drawn at random uniformly among permutations of $\{0, \ldots, 255\}$. This is actually to be expected as it would be quite improbable that a random S-box would exhibit so strong a distinguishing property as having a branch number of three. However, we do not believe that it is possible to exploit this structure in an attack.

4 Implementation of LITTLUN-1

4.1 Hardware implementation

We give a circuit (using OR, AND and XOR gates) implementing the 4-bit S-box underlying LITTLUN-1 in Fig. 4.1. A hardware implementation of the entire S-box can easily be deduced by plugging this circuit into the one of Fig 3.1b. As previously mentioned, LITTLUN-1 can be implemented with 12 non-linear (OR and AND) gates and 24 XOR gates. With a typical cell library such as the Virtual Silicon standard cell library, OR and AND gates cost 1.33 gate equivalent (GE), and XOR gates 2.67 GE. Thus synthesising the S-box with this library would cost 80 GE.

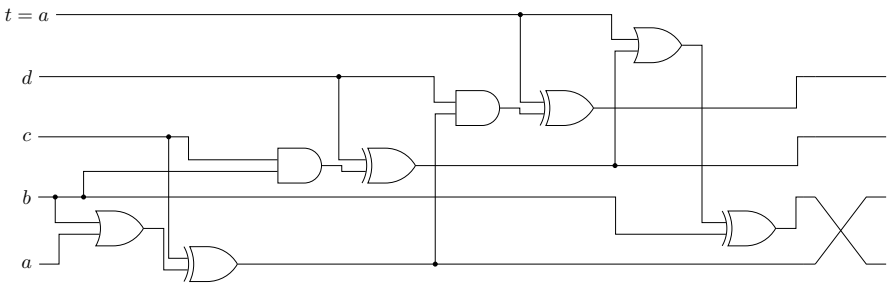


Figure 4.1: A circuit implementing the 4-bit S-box underlying LITTLUN-1 The symbols ⊐, ⊃ and ⋑ represent the AND, OR and XOR gates respectively.

4.2 Bitsliced software implementation

One of our main objective w.r.t. implementation was to obtain an S-box with an efficient *bitsliced* implementation in software. This is closely related to the simplicity of the circuit of the S-box, though not exactly equivalent. We purposefully chose a 4-bit S-box from the *class 13* of Ullrich *et al.* [UDI+11] because of its very efficient bitsliced implementation that requires only 9 instructions on a wide variety of platforms (and offers good instruction-level parallelism). Such an implementation is given in Fig. 4.2. From this, it is easy to obtain an efficient bitsliced implementation for the whole S-box, as shown in Fig. 4.3. This implementation typically requires 43 instructions and 13 registers.

```
t  = b;    b |= a;    b ^= c; // (B): c ^ (a | b)
c &= t;    c ^= d;            // (C): d ^ (c & b)
d &= b;    d ^= a;            // (D): a ^ (d & B)
a |= c;    a ^= t;            // (A): b ^ (a | C)
```

Figure 4.2: Snippet for a bitsliced `C` implementation of the 4-bit S-box with input and output in registers a, b, c, d (the word holding the most significant bit is taken to be a), using one extra register t.

```
t = a ^ e;
u = b ^ f;
v = c ^ g;
w = d ^ h;
S4(t,u,v,w); // uses one more extra register x
a ^= t;     e ^= t;
b ^= u;     f ^= u;
c ^= v;     g ^= v;
d ^= w;     h ^= w;
S4(a,b,c,d); // reuses t as extra
S4(e,f,g,h); // reuses u as extra
```

Figure 4.3: Snippet for a bitsliced `C` implementation of Littlun-1, using the 4-bit S-box of Fig. 4.2 as subroutine. The input and output registers are a, b, c, d, e, f, g, h (with the most significant bit in word a), the four extra registers are t, u, v, w.

4.3 SIMD software implementation

In the context of 4 to 8-bit S-boxes, the Lai-Massey structure of the Littlun construction also allows to conveniently use "vector" Single Instruction Multiple Data (or SIMD) instructions for efficient implementations. We discuss here an implementation of Littlun-1 based mostly on the `pshufb` instruction from Intel's SSSE3 instruction set. This instruction allows to perform 16 parallel lookups of a 4-bit S-box on a 128-bit register, which is particularly convenient to implement a similarly parallel application of a Littlun S-box on 128 bits. More precisely, the semantics of $x' :=$ `pshufb` $x\ y$ can be defined as:

$$x'[i] := \begin{cases} x[\lfloor y[i] \rfloor_4] & \text{if the most significant bit of } y[i] \text{ is not set} \\ 0 & \text{otherwise} \end{cases}$$

where x and y are vectors of 16 bytes. The `pshufb` instruction can easily be used in an implementation either by directly writing the relevant part of the program in assembler or by using compiler intrinsics for a language such as `C`. In the latter case, the intrinsic corresponding to the use of `pshufb` is usually named `_mm_shuffle_epi8`. We give a small function implementing the Littlun-1 S-box using `C` intrinsics in Fig. 4.4. Even without further tuning of the code, this function compares favourably with vector implementations of other S-boxes in terms of efficiency. For instance, it needs about half the number of instructions of Hamburg's hand-written vector implementation of the AES S-box [Ham09], although this must be moderated by the fact that the AES S-box is cryptographically stronger.

4.4 Masking

The low number of non-linear gates needed to implement Littlun-1 makes it a suitable choice for applications where counter-measures against side-channel attacks are required. Indeed, it directly implies a lower cost when using Boolean masking schemes (both hardware and software), which represent the primitive to be masked as a circuit [ISW03, RP10]. In particular, Littlun-1 is competitive with the S-boxes proposed by Grosso *et al.* [GLSV14]: it has the same gate count as the S-box used for Robin and only one more non-linear (and one less XOR) gate than the one used for Fantomas. All three S-boxes are comparable in terms of cryptographic properties.

4.5 Inverse S-box

The inverse Littlun-1^{-1} of Littlun-1 is slightly costlier to implement, because of a more expensive inverse for the underlying 4-bit S-box. As a circuit, the latter requires 5 XOR gates, 4

```
__m128i littlun_ps(__m128i x)
{
  __m128i xlo, xhi, xmid;

  __m128i LO_MASK = _mm_set1_epi8(0x0f);
  __m128i LO_SBOX = _mm_set_epi32(0x01030605, 0x0b090e0d, 0x0802070c, 0x0f040a00);
  __m128i HI_SBOX = _mm_set_epi32(0x10306050, 0xb090e0d0, 0x802070c0, 0xf040a000);

  xhi  = _mm_srli_epi16(x, 4);
  xhi  = _mm_and_si128(xhi, LO_MASK);
  xlo  = _mm_and_si128(x, LO_MASK);
  xmid = _mm_xor_si128(xlo, xhi);

  xmid = _mm_shuffle_epi8(LO_SBOX, xmid);
  xlo  = _mm_xor_si128(xlo, xmid);
  xhi  = _mm_xor_si128(xhi, xmid);

  xlo = _mm_shuffle_epi8(LO_SBOX, xlo);
  xhi = _mm_shuffle_epi8(HI_SBOX, xhi);
  x   = _mm_xor_si128(xlo, xhi);

  return x;
}
```

Figure 4.4: Snippet for an SSE `C` implementation of Littlun-1 using compiler intrinsics.

non-linear (OR and AND) gates and one NOT gate (costing 0.67 GE). The total hardware cost of Littlun-1^{-1} is thus 90 GE.

Software bitsliced implementations are also more expensive. We give a snippet for the inverse of the 4-bit S-box in Fig. 4.5 that requires 11 instructions and 5 registers. The complete inverse can be implemented with 49 instructions and 13 registers in a straightforward adaptation of Fig. 4.3[5].

```
t  = c;    c &= b;    c ^= d; // (A): d ^ (b & c)

d |= t;    d ^= a;            // (B): a ^ (c | d)

a &= c;    a ^= b;    a ^= d; // (C): b ^ B ^ (a & A)

b = ~b;    b &= d;    b ^= t; // (D): c ^ (~b & B)
```

Figure 4.5: Snippet for a bitsliced `C` implementation of the inverse of the 4-bit S-box with inputs in registers a, b, c, d (the word holding the most significant bit is taken to be a), using one extra register t. The output is in c, d, a, b.

5 An application: the Fly block cipher

In this section, we present the Fly block cipher as an application of the Littlun-1 S-box. It is a 64-bit block cipher with 128-bit keys. Thanks to the branch number of the S-box, it is easy to design a round function with good resistance to statistical attacks by combining its bitsliced application with a simple bit permutation. This results in a cipher with a structure similar to Present [BKL+07] with a tradeoff: the S-box is bigger (and thus more expensive to implement, in particular in hardware) but the permutation is simpler (and thus cheaper to implement in software). This cipher was designed to be used in the same cases as Pride, and its chief implementation target is 8-bit microcontrollers.

[5] Because the output registers form a non-trivial permutation of the input ones, additional instructions may also be needed in the cases where this cannot be dealt with implicitly.

5.1 Specifications

We first give the specification of the round function $\mathcal{R}_{\text{FLY}}$ of FLY. It takes a 64-bit block and 64-bit round key as input. Let $x := (x_0||x_1||x_2||x_3||x_4||x_5||x_6||x_7)$, $rk := (rk_0||rk_1||rk_2||rk_3||rk_4||rk_5||rk_6||rk_7)$ be such an input, with x_i, rk_i 8-bit words[6]. Let us define $f_i(t) := \iota(t_0)||t_1||t_2||t_3||t_4||t_5||t_6||t_7$, with $\iota(x) := x + i \mod 256$. We write ARK_i the addition of the i^{th} round key: $\text{ARK}_i(rk_i, x) := f_i(x \oplus rk_i)$, BLS$(x)$ a bitsliced application of LITTLUN-1 (such as *e.g.* the one shown in Fig 4.3) and ROT the "SHIFTROW" word-wise rotation (with $\circlearrowleft$ denoting bitwise rotation to the left)

$$\text{ROT}(x) := (x_0||x_1 \circlearrowleft 1||x_2 \circlearrowleft 2||x_3 \circlearrowleft 3||x_4 \circlearrowleft 4||x_5 \circlearrowleft 5||x_6 \circlearrowleft 6||x_7 \circlearrowleft 7)$$

which can alternatively be defined at the bit level as the permutation $\pi(i) := (i + 8(i \mod 8)) \mod 64$ applied to a suitable binary representation of $x = b_0 \dots b_{63}$. Then we simply have (omitting the addition of the round constant) $\mathcal{R}_{\text{FLY}}(\cdot,\cdot) := \text{ROT} \circ \text{BLS} \circ \text{ARK}$. We give a graphical representation of the SPN structure of this (slightly simplified) round function at the bit level in Fig. 5.1.

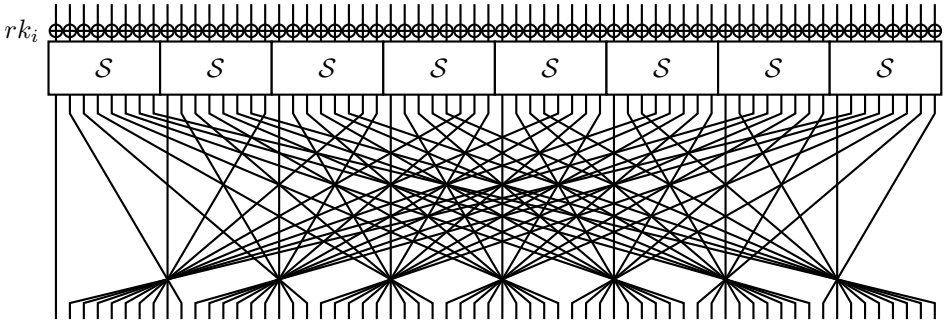


Figure 5.1: The round function of FLY. Bits are numbered left to right from 0 to 63 (w.r.t. the bit permutation). The addition of the round constant is omitted.

We propose two key-schedules, KS1 and KS2, depending on whether resistance to related-key attacks is required (in the case of KS2) or not. In order to distinguish between the two block ciphers, we write FLY for the (default) case where KS1 is used and FLY_{RK} when KS2 is used. We describe KS1 first, which is somewhat similar to the key-schedule of PRIDE by its simplicity. The main difference is that we use both halves k_0 and k_1 of the key $k := k_0||k_1$ to generate all the round keys, whereas PRIDE uses an FX construction where one half of the key is only used for pre- and post-whitening. The latter construction leads to a simpler way to merge on-the-fly round-key generation with the round function, but significantly degrades the security of the cipher to generic attacks from 128 bits to $128 - \log(D)$, with D the amount of data available to an attacker [KR01], while also leading to more efficient time-memory-data trade-offs [Din15].

The sequence (rk) of round keys of KS1 is the simple alternation of k_0 and k_1 defined as $rk_i := k_0 \oplus i \times k_1$[7]. Note however that a round constant is added in ARK through the function f_i.

For FLY to be resistant to related-key attacks, we use the same approach as NOEKEON [DPAR00] to define KS2 as follows. Let us denote by $\text{FLY}(0,\cdot)/12$ twelve applications of the round function of FLY with the all-zero 128-bit key and define $k' := k'_0||k'_1 = \text{FLY}(0, k_0)/12||\text{FLY}(0, k_1)/12$. Then KS2 is defined through FLY_{RK} as $\text{FLY}_{RK}(k,\cdot) := \text{FLY}(k',\cdot)$.

[6]The big endian convention is used to convert from x and rk to the x_is and rk_is.
[7]By writing $i \times k_1$, we mean the all-zero key for even values of i and k_1 otherwise.

The round function of Fly is applied 20 times, the same as Pride. The entire cipher can thus finally be defined as $\text{Fly}(k, \cdot) := \text{Ark}(rk_{20}, \cdot) \circ \mathcal{R}_{\text{Fly}}(rk_{19}, \cdot) \circ \ldots \circ \mathcal{R}_{\text{Fly}}(rk_1, \cdot) \circ \mathcal{R}_{\text{Fly}}(rk_0, \cdot)$.

Design rationale

The core of Fly is the Littlun-1 S-box, which was designed to have a branch number of three. This allows to achieve a good diffusion when combining the S-box application with a simple bit permutation. The latter was chosen so that all eight bits at the output of an S-box go to one different S-box each (similarly, all input bits come from a different S-box). Unlike in Present, this permutation also has cycles of different lengths (discounting fixed points), namely 2 (on 8 values), 4 (on 16) and 8 (on 32). This might reduce the impact of (linear and differential) trail clustering.

The two components of the round function can also be efficiently implemented on an 8-bit architecture through a bitsliced application of the S-box and word rotations respectively (cf. Sec 5.3).

The two key-schedules were designed according to different possible scenarios. Most applications do not require resistance to related-key attacks and a simple alternating key-schedule is enough in that case. We chose not to use an FX construction as in Pride as we did not consider the slight gain in efficiency it offers to be worth the generic security loss it implies. In the spirit of Noekeon, we propose a second key-schedule to offer resistance to related-key attacks that consists in "scrambling" the master key with a permutation of good differential uniformity before it is used as in the first key-schedule.

5.2 Preliminary cryptanalysis

We now analyse the security of Fly against various types of attacks. Considering the similarity of the design with Present and the published analysis on this cipher, the most efficient attacks on Fly are likely to be (variants of) classical statistical (differential and linear) attacks, which we analyse first (in the single-key setting). We then give an overview of the resistance against other attack techniques.

5.2.1 Statistical attacks

We can use the branch number of Littlun-1 together with the properties of the bit permutation Rot to easily derive a lower bound on the number of (differentially and linearly) active S-boxes. Indeed, as the branch number is 3, we are guaranteed to have at least 3 active S-boxes every two rounds of any non-trivial differential or linear characteristic.

The block size of Fly being 64 bits, we want any differential characteristic to have a probability $p \approx 2^{-64}$ when averaged over the key and message space. Similarly, we want any linear characteristic to have an average bias $b \approx 2^{-32}$. From the Prop. 1 of Littlun-1, by multiplying the differential probabilities and applying the piling-up lemma respectively, this means that we want a differential (respectively linear) characteristic to have *at least* 16 active S-boxes. This happens at the latest after 12 rounds, for which at least 18 S-boxes are guaranteed to be active. Even by discounting the additional 2 S-boxes and assuming that a distinguisher can be found for this amount of rounds, this gives a very comfortable margin of 8 rounds, which we estimate to be much beyond the ability of an attacker to convert the distinguisher into, say, key-recovery (in particular, this is twice the number of rounds needed for full diffusion). This also leaves some margin to ensure that even in the case where Fly would exhibit a strong differential or linear hull it would be unlikely for an attacker to be able to mount a meaningful attack. For instance, after 14 rounds, an attacker would need about 2^{20} contributing characteristics to obtain a distinguisher with non-trivial probability, and would still be facing 6 rounds to mount an attack.

Thus, we conjecture that FLY with 20 rounds offers good resistance to statistical attacks.

5.2.2 Other attacks

Algebraic attacks. We would like to estimate how many rounds of FLY are necessary for the degree of the cipher to reach the maximum of 63, as a lower degree could be exploited in "algebraic" attacks. Computing the exact degree of an iterated function is a difficult problem in general, but we can use the upper bound of Boura, Canteaut and De Cannière to estimate how quickly the degree increases [BCD11]. In our case, this bound states that $\deg(G \circ F) \leq n - (n - \deg(G))/(n_0 - 2)$, where n is the block size and n_0 the size of the S-box. Combining this bound with the fact that the degree of the S-box is 5 (and thus that $\deg(\text{BLS} \circ F) \leq 5 \cdot \deg(F)$), we can see that 5 rounds of FLY are necessary to reach a full degree. If we assume this bound to be an equality, any (algebraic) distinguisher on more than about twice this number of rounds is unlikely to exist. Even when relaxing this latter assumption, 20 rounds seem to be well enough to make FLY resistant to algebraic attacks.

Meet-in-the-middle attacks. We analysed how many rounds are necessary to ensure that every bit of the (intermediate) ciphertext depends on every bit of the key, as a basic way to estimate the resistance of FLY to meet-in-the-middle (MitM) attacks, which typically exploit the opposite effect. We did this by performing random trials with 2^{20} pairs of random keys and random plaintexts and found that this happens after at most 5 rounds. Any MitM attack on more than about twice this number of rounds is unlikely to exist, and we therefore conjecture that FLY is resistant to such attacks[8].

Integral attacks, impossible differentials, zero correlation. We did not analyse in detail the security of FLY against integral attacks, nor against impossible differentials and zero correlation attacks. Indeed, none of these techniques seem to be able to attack a significant number of rounds of bit-oriented ciphers such as PRESENT or FLY and we do not consider them to be a threat for our cipher.

Related-key attacks. We now study the resistance of FLY against (XOR-induced, differential) related-key attacks. FLY equipped with the simple key-alternating key-schedule KS1 offers (nearly) no resistance to related-key attacks. With KS2, however, an attacker is unable to control the differences between two different effective master keys $k'_0||k'_1$ and $\widetilde{k'_0||k'_1}$ with a probability much better than $2^{-2 \cdot 64}$, as each difference pair $(k_0, \widetilde{k_0})$ and $(k_1, \widetilde{k_1})$ goes through a permutation with maximum expected differential probability not significantly above 2^{-64}. Furthermore, unlike single-key differential attacks, which introduce differences on the plaintext, we do not expect an attacker to easily be able to force a change of (related) keys if their effective master keys fail to verify a difference relation. Thus, even if a differential on KS2 with probability p higher than 2^{-128} were found, it would only lead to a related-key attack on a weak-key class (of size $\approx p/2^{-128}$) or to an attack requiring a huge amount of keys. Putting everything together, we believe FLY_{RK} to be resistant to XOR-induced related-key attacks.

Known- and chosen-key distinguishers, compression function mode. We do not claim any resistance of FLY against known-key and chosen-key distinguishers. We do not make any claim about its suitability to build a cryptographically strong compression function.

[8] Which key-schedule (KS1 or KS2) is used is irrelevant, as KS2 is equivalent to using KS1 with a different "effective" master key, which a MitM attacker can recover in the exact same way as the "true" master key produced by KS1.

5.3 Implementation

We conclude this discussion on Fly by showing how it can be implemented efficiently on 8-bit AVR microcontrollers. We also argue that the overhead in hardware implementations when compared to a hardware-oriented cipher such as Present is limited.

5.3.1 Microcontrollers implementations

The S-box application can take advantage of the bitsliced expression of Littlun-1 from Sec. 4, which can easily be implemented with instructions available on the cheapest ATtiny chips [Atm07]. It is even possible to save 2 instructions from the 43 quoted in Sec. 4 on higher-end architectures such as the ATmega family [Atm13] by using word-wise 16-bit `movw` instructions, resulting in the implementation given in Fig. B.1 of App. B. A straightforward implementation of the inverse S-box application requires 59 instructions —a significant overhead of 44 %. However, any proper scenario using a lightweight cipher should not need to implement its inverse altogether.

Even though the AVR instruction set does not include rotations by an arbitrary constant, the permutation Rot can still be compactly implemented with only 11 instructions, as shown in Fig. B.2 of App. B.

The entire substitution and permutation layers of Fly can therefore be implemented with only 52 instructions on ATmega (54 on ATtiny), which is 4 less than the 56 of Pride [ADK+14], while at the same time having at least 1.5 times more *equivalent* active S-boxes every two rounds[9].

On-the-fly computation of one round-key of the key-schedule KS1 can be done in 8 instructions. The complete key expansion and addition can be done in 17 instructions as shown in Fig. B.3.

The total round function of Fly including the key-schedule can thus be implemented in 69 instructions. This is one more instruction than Pride, but the conjectured security margin of Fly is much bigger and its resistance to generic attacks does not decrease with the amount of data available to the adversary.

5.3.2 Hardware implementations

We did not implement Fly in hardware, but we can try to estimate the cost (in GE) of such an implementation by looking at the cost of Present. A round-based ASIC implementation of Present-128 can be done for 1884 GE [Pos09], of which 27×16 are dedicated to implementing the 16 S-boxes. If we make the (reasonable) assumption that the key-schedule of Fly does not use significantly more area than the one of Present-128, we can estimate that a similar round-based implementation of Fly would cost in the area of $1884 - 27 \times 16 + 80 \times 8 = 2076$ GE, meaning that the overhead is about 10 %.

References

[ADK+14] Martin R. Albrecht, Benedikt Driessen, Elif Bilge Kavun, Gregor Leander, Christof Paar, and Tolga Yalçin. Block Ciphers - Focus on the Linear Layer (feat. PRIDE). In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2014.

[9] There are at least four active 4-bit S-boxes of maximum differential probability 2^{-2} and best linear correlation 2^{-1} every two rounds of Pride and there are at least 3 active 8-bit S-boxes of maximum differential probability 2^{-4} and best linear correlation 2^{-2} every two rounds of Fly.

[Atm07] Atmel. 8-bit AVR Microcontroller with 1K Byte Flash, Rev. 1006FS-AVR-06/07.

[Atm13] Atmel. 8-bit AVR Microcontroller with 8KBytes In-System Programmable Flash, Rev. 2486AA-AVR-02/2013.

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A New Block Cipher Proposal. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 1998.

[BCD11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-Order Differential Properties of Keccak and *Luffa*. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 252–269. Springer, 2011.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[BR03] Paulo S. L. M. Barreto and Vincent Rijmen. The WHIRLPOOL Hashing Function, May 2003.

[CDL15] Anne Canteaut, Sébastien Duval, and Gaëtan Leurent. Construction of Lightweight S-Boxes using Feistel and MISTY structures (Full Version). *IACR Cryptology ePrint Archive*, 2015:711, 2015.

[Din15] Itai Dinur. Cryptanalytic Time-Memory-Data Tradeoffs for FX-Constructions with Applications to PRINCE and PRIDE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 231–253. Springer, 2015.

[DPAR00] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. The NOEKEON Block Cipher. Nessie Proposal, October 2000.

[GLSV14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2014.

[Ham09] Mike Hamburg. Accelerating AES with Vector Permute Instructions. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2009.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[JV04] Pascal Junod and Serge Vaudenay. FOX : A New Family of Block Ciphers. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, volume 3357 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 2004.

[KR01] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search (an Analysis of DESX). *J. Cryptology*, 14(1):17–35, 2001.

[LM91] Xuejia Lai and James L. Massey. Markov Ciphers and Differential Cryptanalysis. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991.

[LP07] Gregor Leander and Axel Poschmann. On the Classification of 4 Bit S-Boxes. In Claude Carlet and Berk Sunar, editors, *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007.

[Nat01] National Institute of Standards and Technology. FIPS 197: Advanced Encryption Standard (AES), November 2001.

[Pos09] Axel Poschmann. *Lightweight Cryptography*. PhD thesis, Ruhr-Universität Bochum, 2009.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.

[UDI+11] Markus Ullrich, Christophe De Cannière, Sebastian Indesteege, Özgül Küçük, Nicky Mouha, and Bart Preneel. Finding Optimal Bitsliced Implementations of 4×4-bit S-boxes. In *Symmetric Key Encryption Workshop, SKEW 2011, Lyngby, Denmark, February 16-17 2011.*, 2011.

[Vau99] Serge Vaudenay. On the Lai-Massey Scheme. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *Advances in Cryptology - ASIACRYPT '99, International Conference on the Theory and Applications of Cryptology and Information Security, Singapore, November 14-18, 1999, Proceedings*, volume 1716 of *Lecture Notes in Computer Science*, pages 8–19. Springer, 1999.

[ZBL+14] Wentao Zhang, Zhenzhen Bao, Dongdai Lin, Vincent Rijmen, Bohan Yang, and Ingrid Verbauwhede. RECTANGLE: A Bit-slice Ultra-Lightweight Block Cipher Suitable for Multiple Platforms. *IACR Cryptology ePrint Archive*, 2014:84, 2014.

[ZWWD14] Jingyuan Zhao, Xiaoyun Wang, Meiqin Wang, and Xiaoyang Dong. Differential Analysis on Block Cipher PRIDE. *IACR Cryptology ePrint Archive*, 2014:525, 2014.

A Complement on the properties of LITTLUN-1

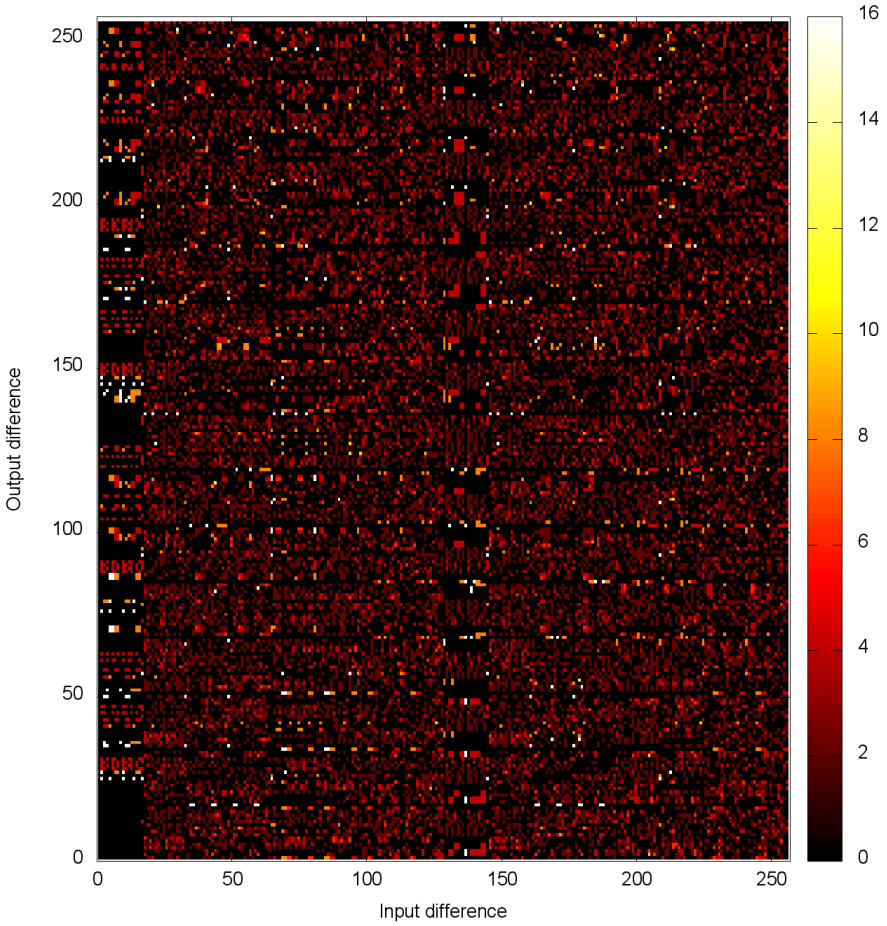


Figure A.1: DDT of LITTLUN-1

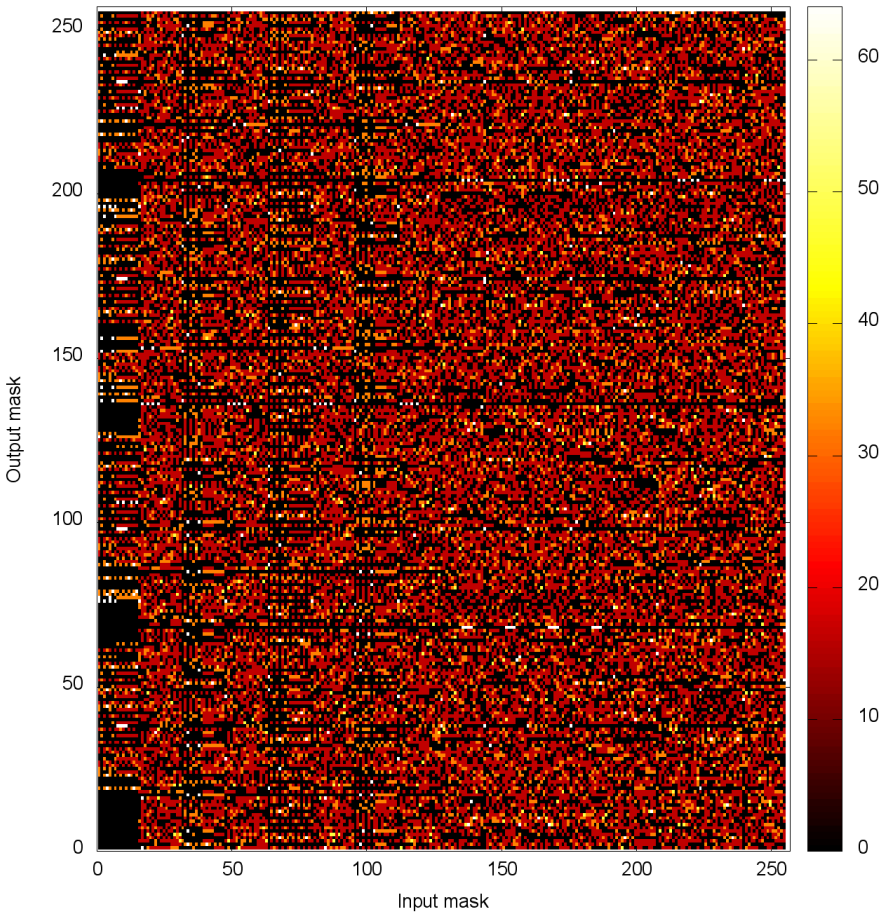


Figure A.2: LAT of Littlun-1

B AVR implementation of the FLY round function

We give pseudo AVR assembly code for the S-box layer, the permutation and on-the-fly computation of the key-schedule of FLY.

```
; input/output in s0,...,s7 (with MSBs in s0)
; temporary values held in t0,...,t4
; top XOR
movw t0, s0 ; moves t1:s0 <- s1:s0
movw t2, s2
eor t0, s4
eor t1, s5
eor t2, s6
eor t3, s7
; middle S-box
mov t4, t1
or  t1, t0
eor t1, t2
and t2, t4
eor t2, t3
and t3, t1
eor t3, t0
or  t0, t2
eor t0, t4
; bottom XOR
eor s0, t0
eor s1, t1
eor s2, t2
eor s3, t3
eor s4, t0
eor s5, t1
eor s6, t2
eor s7, t3
; bottom S-boxes
mov t0, s1
or  s1, s0
eor s1, s2
and s2, t0
eor s2, s3
and s3, s1
eor s3, s0
or  s0, s2
eor s0, t0

mov t0, s5
or  s5, s4
eor s5, s6
and s6, t0
eor s6, s7
and s7, s5
eor s7, s4
or  s4, s6
eor s4, t0
```

Figure B.1: The LITTLUN-1 S-box on ATmega, using 41 instructions.

```
; input/output in s0,...,s7
rol s1
rol s2
rol s2
swap s3
ror s3
swap s4
swap s5
rol s5
ror s6
ror s6
ror s7
```

Figure B.2: The ROT permutation on ATmega/ATtiny, using 11 instructions.

```
; key input in k0,...,k15
; cipher state in s0,...,s7
; round counter in c0
eor k8, k0
eor k9, k1
eor k10, k2
eor k11, k3
eor k12, k4
eor k13, k5
eor k14, k6
eor k15, k7

eor s0, k8
eor s1, k9
eor s2, k10
eor s3, k11
eor s4, k12
eor s5, k13
eor s6, k14
eor s7, k15

add s0, c0
```

Figure B.3: The KS1 key-schedule on ATmega/ATtiny, using 17 instructions.

C Test vectors for FLY

All numbers are given in big endian.

```
k0: 0x0000000000000000 k1: 0x0000000000000000
p : 0x0000000000000000
FLY(k0||k1,p)             : 0xBC73EF592E56FECC

k0: 0x0001020304050607 k1: 0x08090A0B0C0D0E0F
p : 0xF7E6D5C4B3A29180
FLY(k0||k1,p)             : 0x8AA1CEE6100013D5

k0: 0x0000000000000000 k1: 0x0000000000000000
p : 0x0000000000000000
FLY_RK(k0||k1,p)          : 0x148DC9F9CC65DB64

k0: 0x0001020304050607 k1: 0x08090A0B0C0D0E0F
p : 0xF7E6D5C4B3A29180
FLY_RK(k0||k1,p)          : 0xC73FE2DED9CF5D3C
```