The LITTLUN S-box and the FLY block cipher

Pierre Karpman[a] and Benjamin Grégoire[b]

[a]Centrum Wiskunde & Informatica, The Netherlands
École polytechnique, France
Nanyang Technological University, Singapore

[b]Inria, France

Lightweight Cryptography Workshop, Gaithersburg
2016–10–18

Let's get started

What do we want? Block ciphers!

- Fundamental primitives in (secret-key) cryptography
- Useful to provide confidentiality and/or authenticity

Where do we want them?

- High-end 64-bit processors
- High-end SoC
- Low-end microcontrollers (8, 16, 32 bits)
- (Small) hardware

AES is good!

- AES/Rijndael128, winner of the AES competition (2000)
- 128-bit blocks, {128,192,256}-bit keys
- Fast & versatile
- Good security
- But is AES all what you need?

AES-128 performance on constraint devices

- *Serial* implementation of AES: ≈ 2400 GE (Moradi et al., 2011) (226 cyc. per block)
- On 8-bit microcontroller:
  - 146 cpb, (970 B ROM + 18 RAM) (NSA, 2014)
  - 125 cpb (1912 B ROM + 432 B RAM) (Osvik et al., 2010; Osvik, 2014)
- Not bad at all, but can do (slightly better)
- Lightweight crypto: try to do better than AES in some specific situations (not easy)

Some lightweight block ciphers (academic)

- PRESENT-128 (64-bit block, 128-bit key) (Bogdanov et al., 2007)
  - Round-based implementation: 1884 GE (Poschmann, 2009) (Serial: 1391)
  - Not efficient in software
- PRIDE (64-bit block, 64-bit key + 64-bit for whitening) (Albrecht et al., 2014)
  - On 8-bit microcontrollers, 189 cpb (266 B ROM)

Some lightweight block ciphers (NSA)

Two members in a big family: SIMON and SPECK (NSA, 2013)

- Many possible block & key sizes
- Efficient both in hardware and software
- SPECK64-128 on 8-bit microcontrollers
  - 154 cpb (218 B ROM) (NSA, 2015)
  - 122 cpb (628 B ROM + 108 B RAM) (NSA, 2015)
- SIMON64-128 on 8-bit microcontrollers
  - 290 cpb (253 B ROM) (NSA, 2015)
  - 221 cpb (436 B ROM + 176 B RAM) (NSA, 2015)

Our goal for today

- Design a block cipher (64-bit blocks, 128-bit keys) with good 8-bit implementation
- Roughly comparable with SPECK/PRIDE/SIMON for efficiency
- With easy arguments v. statistical attacks (like PRIDE)
- With efficient countermeasures v. side-channel attacks (like SIMON)
- Conceptually simple

How to do that

- Use a pure SPN structure (like e.g. PRESENT)
- Combine properties of the S and P layer to count active S-boxes (good for security)
- Use a bitsliced S-box and a "rotation" permutation (good for implementation)

A strategy for pure SPNs (1)

Branch number of an S-box

The diff. branch number of an S-box $\mathcal{S}$ is:

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

The lin. branch number of an S-box $\mathcal{S}$ is:

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

- Reminiscent of the B.N. of a linear mapping ($\approx$ min. distance of a linear code)

A strategy for pure SPNs (2)

1. Find an S-box with high diff/lin B.N.
2. Find a bit permutation with "good" diffusion
3. Derive a lower bound on # of active S-boxes

Let's do this: design criteria for an 8-bit S-box

- Diff. & lin. branch number ≥ 3
- MDP $\leq 2^{-4}$, linearity $\leq 2^6$ ($\equiv$ linear bias $\leq 2^{-3}$)
- Efficient bitsliced implementation
- Low overall number of operations

Strategy:

- Start from a "nice" 4-bit S-box
- Use a $2 \times 4 \rightarrow 8$ construction (Feistel, Misty, Lai-Massey, ...)

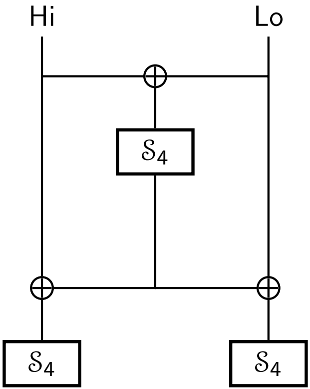
Lai-Massey structure for S-boxes

- Makes 3 calls to the 4-bit S-box with depth 2
- MDP & linarity of the 8-bit S-box ≈ square the one on 4-bit
- 4-bit S-box has Diff. B.N. 3 ⇒ 8-bit S-box has Diff. B.N. 3
- Efficient vector implementations with SSSE3 (not so useful here)

✧

- Condition on Diff. B.N. on 4-bit not necessary
- Lin. B.N. on 8-bit *may be* 3 (not possible for good 4-bit)

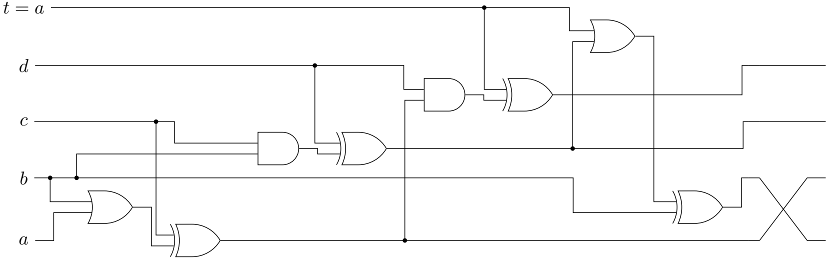
Lai-Massey in a picture



How to instantiate the 4-bit S-box?

- Initial strategy: use fastest SERPENT S-box (has B.N. 3) (Biham et al., 1998)
- In the end: use member of Class 13 (Ulrich et al., 2011)
  - Not B.N. 3 but ⇒ B.N. 3 on 8-bit anyway
  - Min. # of L. and N.L. gates possible for an optimal 4-bit (4 each)
  - Very efficient bitsliced implementations

"LITTLUN-S4" in a picture



Bitsliced implementation of LITTLUN-S4

```
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)
```

- 9 instructions w. 5 registers

Bitsliced implementation of the 8-bit S-box "LITTLUN1"

```
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
```

- 43 instructions w. 13 registers

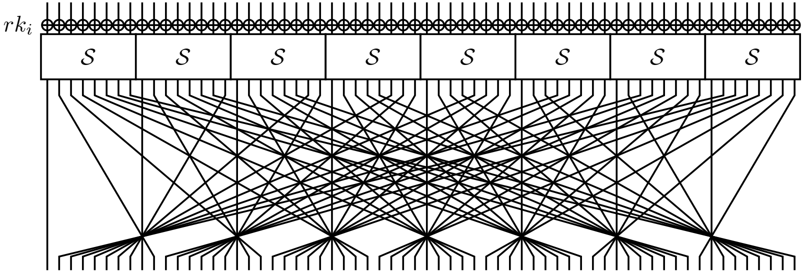
So we are done

- LITTLUN1 meets all the criteria
- Only downside: its inverse is more expensive in bitsliced form (59 inst. v. 43)
  - But we know good inverse-free (authenticated) modes of operation (e.g. CLOC, OTR)

A simple design

- 64-bit blocks, 128-bit key
- Round function, optimized for 8-bit microcontrollers:
  1. Apply LITTLUN1 in bitsliced form to $X_0, X_1, \ldots X_7$ (eight 8-bit words)
  2. Rotate X_i by i to the left
- 20 rounds for the full cipher
- Two key schedules (elementary v. RKA-resistant) (could be improved)

The FLY round function in a picture



Security analysis

- Permutation diffuses "optimally"
- From the B.N. of the S-box ⇒ at least 6 active S-boxes every 4 rounds
- ⇒ at least 18 active S-boxes for 12 rounds ⇒ no single trail with high prob./bias expected
- Other attacks (MiTM, algebraic, integral, impossible diff.) less a concern

Implementation on AVR

- Entire round function + on-the-fly simple key schedule = 76 inst. on ATmega
- 8 more than PRIDE, but with 1.5× more (eqv.) active S-boxes
- ⇒ ≈ 200 cpb., small code (complete perfs. on AVR TBD)

Round function assembly (S-box application)

```
; /S/
movw t0, s0
movw t2, s2
eor t0, s4
eor t1, s5
eor t2, s6
eor t3, s7

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
```

```
eor s0, t0
eor s1, t1
eor s2, t2
eor s3, t3
eor s4, t0
eor s5, t1
eor s6, t2
eor s7, t3

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
```

Round function assembly (Bit permutation)

```
; /P/
rol s1
rol s2
rol s2
swap s3
ror s3
swap s4
swap s5
rol s5
ror s6
ror s6
ror s7
```

Round function assembly (Key application & update)

```
; /ARK/
eor s0, k0
eor s1, k1
eor s2, k2
eor s3, k3
eor s4, k4
eor s5, k5
eor s6, k6
eor s7, k7

eor s0, c0
eor s1, 255
```

```
eor k0, k8
eor k1, k9
eor k2, k10
eor k3, k11
eor k4, k12
eor k5, k13
eor k6, k14
eor k7, k15
```

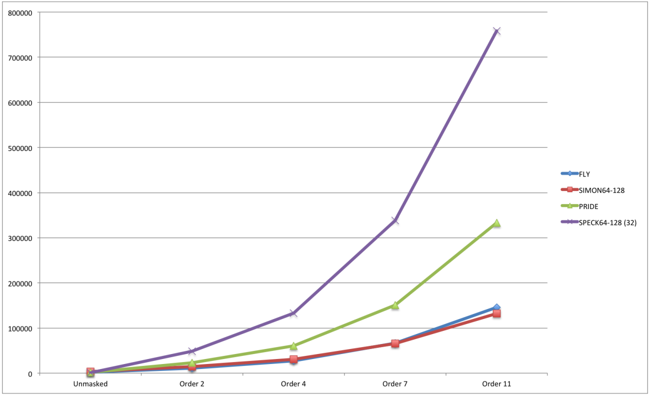
```
mov  t0, c0
andi t0, 1
dec  t0
andi t0, 177
lsr  c0
eor  c0, t0
```

The cost of protection

- Intented implementation target is prone to SCA
- ⇒ should also consider the cost of countermeasures v. e.g. DPA
- We use the masking compiler of Barthe et al. to obtain masked implementation at various orders (2015)
- Comparison with SIMON/SPECK/PRIDE is favourable

Masking cost at various orders

- Generate masked implementation, count #operations to encrypt one block (rough measure)



Conclusion

- LITTLUN1 is a cheap S-box with good diffusion properties
- It is well-suited to a pure SPN design on 64-bit blocks

- FLY is a bitsliced cipher targeting 8-bit microcontrollers
- One of the few bitsliced ciphers with simple security arguments

- Compact and efficient w. or w/o. masking

Fin!