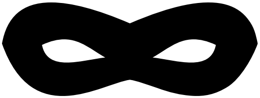
LS-Designs

Bitslice Encryption for Efficient Masked Software Implementations

Vincent Grosso[1] Gaëtan Leurent[1,2]
François-Xavier Standert[1] Kerem Varici[1]

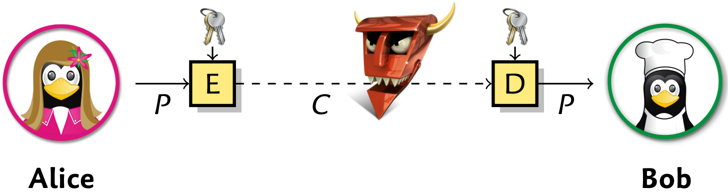
[1]UCL, Belgium & [2]Inria, France

FSE 2014



Secure communications

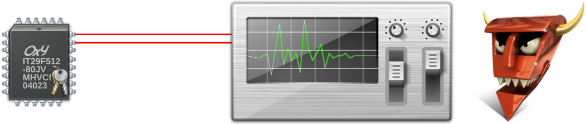
- Cryptography aims to provide secure communications in the presence of an adversary.
- Classical model: adversary controls the communication channel:



- Recovering the plaintext without the key should be hard.
  - Mathematical properties of the cipher *E*.

Side-channel analysis

- In practice, the cryptography is implemented by a physical system
  - Smart card (credit card, SIM), computer, mechanical machine ...
- The adversary can measure physical properties of the system
  - Time to encrypt data
  - Power consumption
  - Electromagnetic radiations
  - Sound
  - ...



- Information about values during the computation can break the system even if the algorithm is good.

Side-channel protection

- Implement crypto carefully:
  - Constant time operations (avoid SPA attacks)
  - No secret branches
  - No secret table access (avoid cache timing)
- Power consumption depend on the value of the operands
  - Correlated with Hamming weight/distance of values in bus/registers/...
  - Exploited in DPA attacks
- Masking
  - Best understood countermeasure

Side-channel protection

- Implement crypto carefully:
  - Constant time operations (avoid SPA attacks)
  - No secret branches
  - No secret table access (avoid cache timing)
- Power consumption depend on the value of the operands
  - Correlated with Hamming weight/distance of values in bus/registers/...
  - Exploited in DPA attacks
- Masking
  - Best understood countermeasure

Side-channel protection

- Implement crypto carefully:
  - Constant time operations (avoid SPA attacks)
  - No secret branches
  - No secret table access (avoid cache timing)
- Power consumption depend on the value of the operands
  - Correlated with Hamming weight/distance of values in bus/registers/...
  - Exploited in DPA attacks
- Masking
  - Best understood countermeasure

Masking

- Split the sensitive data in r shares (secret sharing)
  - $k_1 \leftarrow \$, \ldots$
  - $k_{r-1} \leftarrow \$$
  - $k_r \leftarrow k - \sum k_i$
- Use MPC-like techniques to avoid manipulating the secret itself
  - Linear operations are easy
    - Perform operation on each share
  - Non-linear operations are expansive
    - Need interaction, and randomness
    - Cost increase with r^2
- Side-channel adversary must combine r measures (for an ideal implementation...)
  - Data complexity is exponential in r: $(\sigma_n^2)^r$

Motivation

Main question

How to have secure crypto on 8-bit micro-controllers?

- Side-channel resistance necessary in many lightweight settings
  - Avoid your car keys / credit card being cloned
- Usual approach:
  1. Design a secure cipher (AES, PRESENT, Noekeon, ...)
  2. Implement with side-channel countermeasures
- Can we reverse the problem?
  1. Use operations that are easy to mask
  2. In order to design a secure cipher
- Previous work: Zorro, PICARO

Choice of operations

Important remark

Logic gates are easier to mask than table-based S-boxes
(If we target Boolean masking)

- Use bitsliced S-boxes (SERPENT, Noekeon, ...)
  - One word contains the msb (resp. 2nd bit, ...) of every S-box
  - Bitwise operations: 8 S-boxes in parallel using 8-bit words
  - Use a small number of non-linear gates
- We can use tables for the diffusion layer!
  - Efficient, good diffusion
  - Easy to mask (linear)

Choice of operations

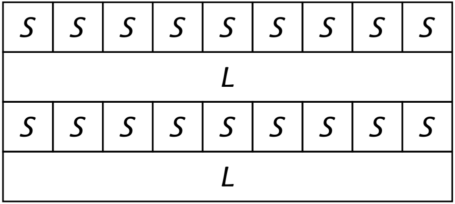
Important remark

Logic gates are easier to mask than table-based S-boxes
(If we target Boolean masking)

- Use bitsliced S-boxes (SERPENT, Noekeon, ...)
  - One word contains the msb (resp. 2^{nd} bit, ...) of every S-box
  - Bitwise operations: 8 S-boxes in parallel using 8-bit words
  - Use a small number of non-linear gates
- We can use tables for the diffusion layer!
  - Efficient, good diffusion
  - Easy to mask (linear)

LS-designs

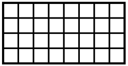
- Mathematical description: SPN network
  - S-boxes (with simple gate representation)
  - Linear diffusion layer (binary matrix)
  - Good design criterion: wide-trail



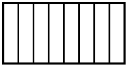
- Bitslice implementation:
  - S-box as a series of bitwise operations
  - L-box tables for diffusion layer
  - Easy to mask (simple non-linear ops., complex linear ops.)

LS-designs

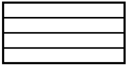
$x \leftarrow P \oplus K$
for $0 \leq r < N_r$ **do**
  ▷ S-box layer:
  for $0 \leq i < l$ **do**
    $x[i, \star] = S[x[i, \star]]$
  ▷ L-box layer:
  for $0 \leq j < s$ **do**
    $x[\star, j] = L[x[\star, j]]$
  ▷ Key addition:
  $x \leftarrow x \oplus k_r$
return x



State as a bit-matrix



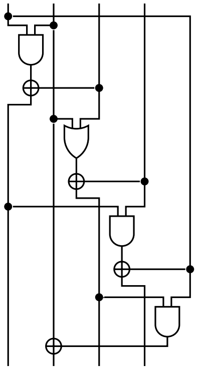
S-box layer



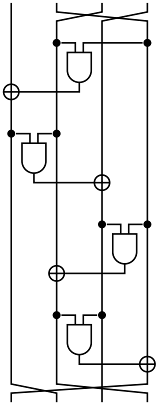
L-box layer

S-box: 4-bit

- Exhaustive search possible for 4-bit Sbox [UCIKMP11]
- Optimal S-box with 4 non-linear gates: $\Pr_{\text{lin}} = 2^{-1}$, $\Pr_{\text{diff}} = 2^{-2}$



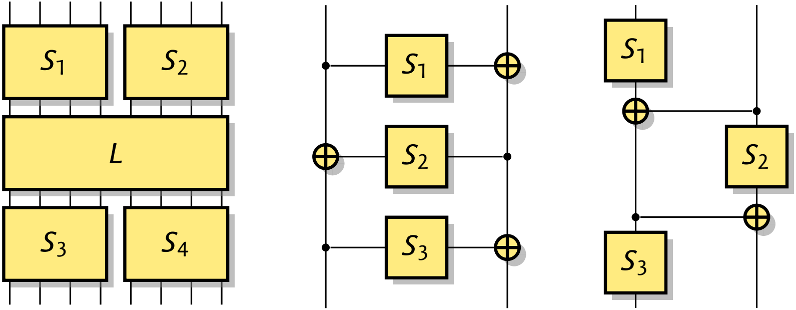
Class13 from [UCIKMP11]



Involution with same prob.

S-box: 8-bit

- Exhaustive search not possible
- Use constructions from a 4-bit S-box:



Whirlpool-like Feistel MISTY-like

- Test properties

Best S-Boxes

	size	#AND	#XOR	Invol.	deg(S)	Pr_{diff}	Pr_{lin}
NOEKEON	4	4	7	Yes	3	2^{-2}	2^{-1}
Class 13		4	4	No	3	2^{-2}	2^{-1}
Figure (b)		4	4	Yes	3	2^{-2}	2^{-1}
AES	8	32	83	No	7	2^{-6}	2^{-3}
Whirlpool + Class 13		16	41	No	6	$2^{-4.68}$	2^{-2}
Whirlpool + Figure (b)		16	42	No	6	$2^{-4.68}$	2^{-2}
Feistel + Class13		12	24	Yes	6	2^{-4}	2^{-2}
Feistel + Figure (b)		12	24	Yes	5	2^{-4}	2^{-2}
MISTY + 3/5-bit		11	25	No	5	2^{-4}	2^{-2}
Feistel^2 + Class13	16	36	96	Yes	13	2^{-8}	2^{-4}

Best S-Boxes

	size	#AND	#XOR	Invol.	deg(S)	Pr_{diff}	Pr_{lin}
NOEKEON	4	4	7	Yes	3	2^{-2}	2^{-1}
Class 13		4	4	No	3	2^{-2}	2^{-1}
Figure (b)		4	4	Yes	3	2^{-2}	2^{-1}
AES	8	32	83	No	7	2^{-6}	2^{-3}
Whirlpool + Class 13		16	41	No	6	$2^{-4.68}$	2^{-2}
Whirlpool + Figure (b)		16	42	No	6	$2^{-4.68}$	2^{-2}
Feistel + Class13		12	24	Yes	6	2^{-4}	2^{-2}
Feistel + Figure (b)		12	24	Yes	5	2^{-4}	2^{-2}
MISTY + 3/5-bit		11	25	No	5	2^{-4}	2^{-2}
Feistel^2 + Class13	16	36	96	Yes	13	2^{-8}	2^{-4}

L-box choice

- Wide trail strategy: maximum branch number
  - At least $\mathcal{B}$ active S-boxes every two rounds
  - Use coding theory results

8-bit Exhaustive search possible

- Maximum branch number is 5
- Reachable with involutions

16-bit Optimal codes known

- Optimal distance is 8
- Reed-Muller(2,5) gives an involution

32-bit Optimal codes not known

- Best known code have a distance 12
- Upper bound is 16

Which S-box with which L-box?

- We want to design a 128-bit cipher
- Compare implementation cost with best trail $\leq 2^{-128}$

- *8-bit L-box, 16-bit S-box*
  At least 16 active S-boxes, *i.e.* 6 rounds
  984 operations: 216 non-linear, 672 linear, 96 table-lookups
- *16-bit L-box, 8-bit S-box*
  At least 32 active S-boxes, *i.e.* 8 rounds
  1088 operations: 192 non-linear, 640 linear, 256 table-lookups
- *32-bit L-box, 4-bit S-box*
  At least 64 active S-boxes, *i.e.* 12 rounds
  1920 operations: 192 non-linear, 960 linear, 768 table-lookups

- Best trade-off: 16-bit L-box, 8-bit S-box
  - Further analysis allows to decrease the number of rounds

Product states

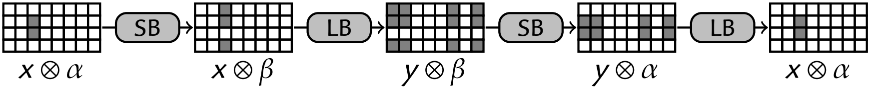
- Special states can be written as a tensor product:

$$\alpha \otimes x = \begin{bmatrix} \alpha_0 x_0 & \alpha_0 x_1 & \alpha_0 x_2 & \alpha_0 x_3 & \cdots & \alpha_0 x_l \\ \alpha_1 x_0 & \alpha_1 x_1 & \alpha_1 x_2 & \alpha_1 x_3 & & \alpha_1 x_l \\ \vdots & & & \vdots & \ddots & \vdots \\ \alpha_s x_0 & \alpha_s x_1 & \alpha_s x_2 & \alpha_s x_3 & \cdots & \alpha_s x_l \end{bmatrix}$$

  - All active S-boxes have the same input α
  - All active L-boxes have the same input x

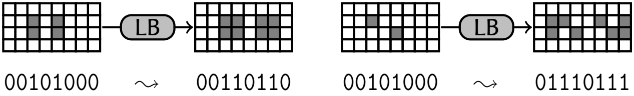
- $S\text{-}layer(\alpha \otimes x) = S(\alpha) \otimes x$, $L\text{-}layer(\alpha \otimes x) = \alpha \otimes L(x)$.

- If components are involutive, product trails are iterative, optimal:



Non-involutive L-box

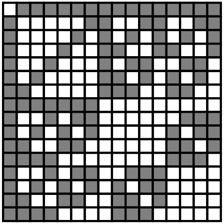
- With non-involutive L-box, no obvious trails reach the bound
- For a given L-box, we run a search for optimal trails:
  1. Consider truncated trails (active/non-active S-boxes)
  2. Compute all possible transitions for the L-layer
     - Including non-linear transitions, *e.g.*



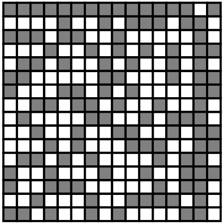
  3. Search shortest paths in the graph
     - l-bit state
     - weighted with number of active S-boxes
     - Feasible for $l \leq 16$
- We use random permutations of a known good code

Non-involutive L-box

- The best L-box we found allow to reduce the number of rounds:



Involutive



Non-involutive

Number of active S-boxes

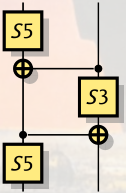
Rounds	1	2	3	4	5	6	7	8	9	10	11	12
Involutive	1	8	9	16	17	24	25	32	33	40	41	48
Non-inv.	1	8	12	20	24	30	34	40	46	52	58	64
AES	1	5	9	25	26	30	34	50	51	55	59	75

Instances

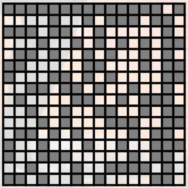
Instances

FANTOMAS

- 128-bit block, 128-bit key
- $k_i = K \oplus c_i$
- Non-involutive components
- 12 rounds



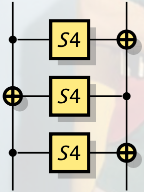
S-box



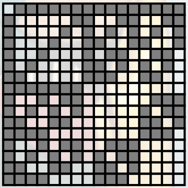
L-box

ROBIN

- 128-bit block, 128-bit key
- $k_i = K \oplus c_i$
- Involutive components
- 16 rounds

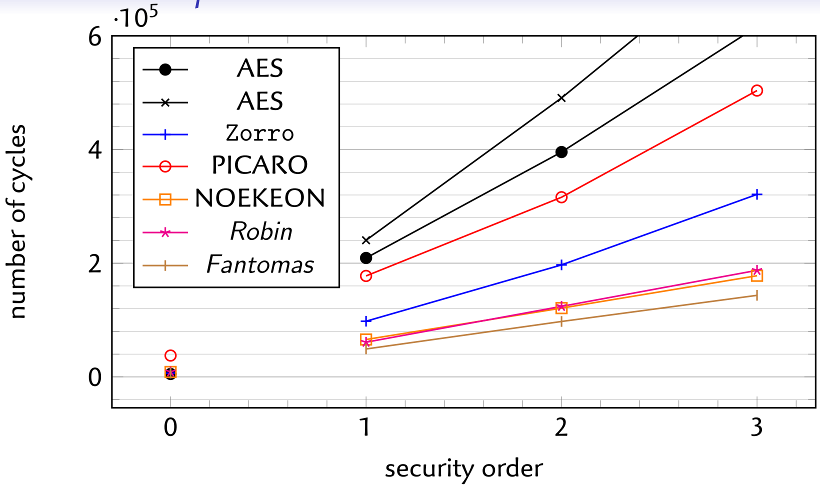


S-box



L-box

Implementation: AVR micro-controller



- Very good performances for masked implementations
- Noekeon also very good (similar components)

Implementation: High-end CPUs

- Also efficient on high-end CPUs with vector engines
- Use large registers (128-bit) for bitsliced S-box
- Use vector permute instructions for L-box
  - 4-bit to 8-bit table with `pshufb` in SSSE3, `vtbl` in NEON
  - 16-bit to 16-bit table as 8 small tables
  - Constant time (no cache timing side-channel)

	Fantomas	*Robin*	AES	
			w/o AES-NI	w/ AES-NI
ARM Cortex A15	14.2	18.1	17.8	N/A
Atom	33.3	43.5	17	N/A
Core i7 Nehalem	6.3	8.1	6.9	N/A
Core i7 Ivy Bridge	4.2	5.5	5.4	1.3

Conclusion

LS-designs

- Bitslice S-box easy to mask
- L-box: table-based linear layer for good diffusion

- Simple and regular SPN structure
  - Avoid irregularities of Zorro
  - Bound for differential/linear trails (wide trail)
- Efficient, easy to mask
  - Good performances for masked implementations
  - Good performances on high-end CPUs
- Future work:
  - Better S-box?
  - Consider related-key attacks
  - CAESAR submission?

Simple Code (16-bit)

```
void C13(uint16_t X[4], uint16_t Y[4]) {
  uint16_t a, b, c, d;
  Y[0] ^= a = (X[0] & X[1]) ^ X[2];
  Y[2] ^= c = (X[1] | X[2]) ^ X[3];
  Y[3] ^= d = (  a  & X[3]) ^ X[0];
  Y[1] ^= b = (  c  & X[0]) ^ X[1];
}
#define Sbox(x) C13(x+4, x), C13(x, x+4), C13(x+4, x)
extern uint16_t L1[256], L2[256];

void Encrypt(uint16_t x[8], uint16_t k[8]) {
  for (int j=0; j<8; j++) x[j] ^= k[j];          // Initial key adition
  for (int i=0; i<16; i++) {
    x[0] ^= L1[i+1];                             // Round constant
    Sbox(x);                                     // S-box
    for (int j=0; j<8; j++) {
      x[j]  = L2[x[j]>>8] ^ L1[x[j]&0xff];       // L-box
      x[j] ^= k[j];                              // Key adition
    }
  }
}
```