



Lightweight Cryptography on ARM

Software implementation of block ciphers

Rafael J. Cruz, Tiago B. Reis, **Diego F. Aranha**, Julio López, Harsh Kupwade Patil

University of Campinas, LG Electronics Inc.

Introduction

Summary

Block ciphers and MACs are among the most fundamental and useful cryptographic primitives.

We discuss techniques for efficient and secure implementations of lightweight block ciphers and MACs in software:

1. FANTOMAS, an LS-Design proposed in [GLSV14].
2. PRESENT, a Substitution-Permutation Network (SPN) [BKL⁺07]
3. ARX-based MACs SIPHASH and CHASKEY.

We target **low-end** and **NEON-capable ARM** processors, typical of embedded systems. Results are part of a project sponsored by LG involving more than 20 symmetric/asymmetric algorithms.

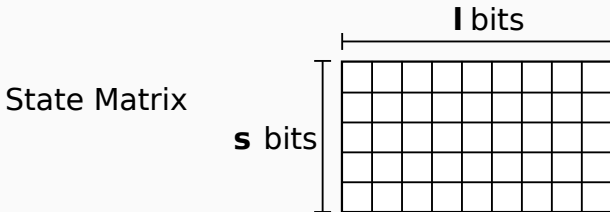
Fantomas

Construction

LS-Designs

Paradigm to construct block ciphers providing:

- **Lightweight** designs from simple substitution and linear layers.
- Friendliness to **side-channel countermeasures** (*bitslicing* and *masking*).
- Tweakable variant for **authenticated encryption** (SCREAMv3).



Construction

Algorithm 1 LS-Design encrypting block B into ciphertext C with key K .

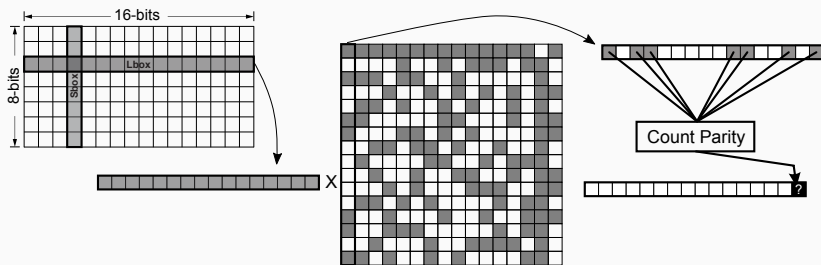
```
1:  $C \leftarrow B \oplus K$  ▷  $C$  represents an  $s \times l$ -bit matrix
2: for  $0 \leq r < N_r$  do
3:   for  $0 \leq i < l$  do ▷ S-box layer
4:      $C[i, \star] = S[C[i, \star]]$ 
5:   end for
6:   for  $0 \leq j < s$  do ▷ L-box layer
7:      $C[\star, j] = L[C[\star, j]]$ 
8:   end for
9:    $C \leftarrow C \oplus K \oplus C(r)$  ▷ Key and round constant addition
10: end for
11: return  $C$ 
```

Algorithm

The LS-Design paper introduced an **involutive** instance (Robin), and a **non-involutive** cipher (Fantomas).

Fantomas

- **128-bit** key length and block size.
- **No** key scheduling.
- 8-bit (3/5-bit 3-round) **S-boxes** from *MISTY*.
- **L-box** from *vector-matrix product* in \mathbb{F}_2 .



Implementation in 32/64 bits

Internal state can be represented with union to respect **strict aliasing rules** for 16/32/64-bit operations:

```
typedef union {  
    uint32_t u32;    // uint64_t u64;  
    uint16_t u16[2]; // uint16_t u16[4];  
} U32_t;
```

S-boxes operate over **16-bit chunks** in the **u16** portion.

Key addition works using the **u32/u64** internal state:

```
for (j=0; j < 4; j++)    // for(j=0; j < 2; j++)  
    st[j].u32 ^= key_32[j]; //    st[j].u64 ^= key_64[j];
```


Implementation in 32/64 bits

L-box can be evaluated using **two precomputed tables**:

```
/* Unprotected L-box version */  
st[j].u16[0] = LBoxH[st[j].u16[0]>>8] ^  
               LBoxL[st[j].u16[0] & 0xff];  
st[j].u16[1] = LBoxH[st[j].u16[1]>>8] ^  
               LBoxL[st[j].u16[1] & 0xff];
```

Problem: Beware of cache-timing-attacks!

Implementation in 32/64 bits

L-box can be evaluated using **two precomputed tables**:

```
/* Unprotected L-box version */  
st[j].u16[0] = LBoxH[st[j].u16[0]>>8] ^  
               LBoxL[st[j].u16[0] & 0xff];  
st[j].u16[1] = LBoxH[st[j].u16[1]>>8] ^  
               LBoxL[st[j].u16[1] & 0xff];
```

Problem: Beware of cache-timing-attacks!

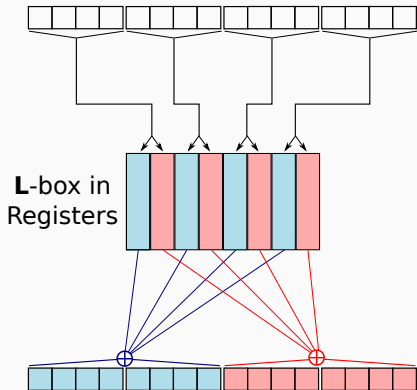
Attacker who monitors **L-box positions in cache** can recover internal state. Internal state trivially reveals **keys and plaintext** if recovered right before/after last/first key addition.

Solution: We can replace memory access with online computation:

```
static inline type_t LBox(type_t x, type_t y, uint8_t s) {
    x &= y;
    x ^= x >> 8;
    x ^= x >> 4;
    x ^= x >> 2;
    x ^= x >> 1;
    return (x & 0x00010001) << s;
    // return (x & 0x0001000100010001) << s;
}
```

NEON implementation

L-boxes can be evaluated using **shuffling** instructions to compute **8 table lookups** in parallel.

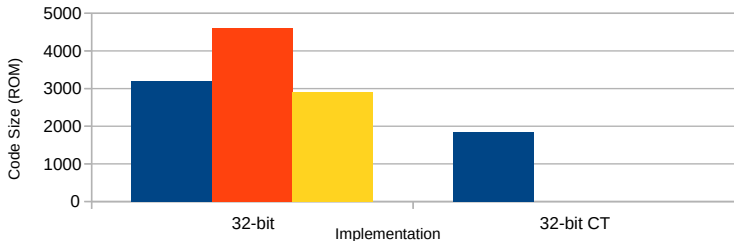
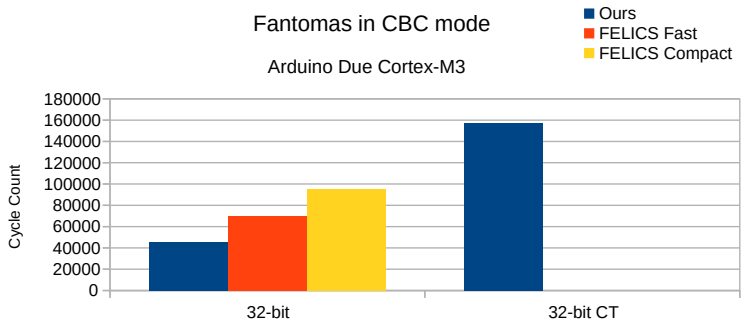


Important: 32-bit implementations can process 2 blocks and vector implementations can process **16 blocks** simultaneously in CTR mode.

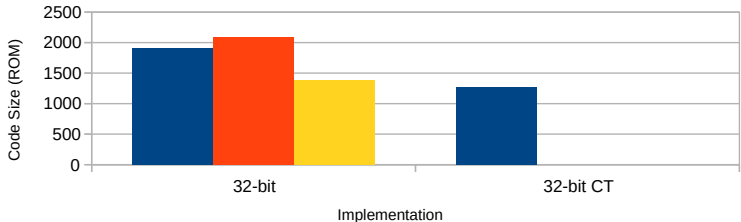
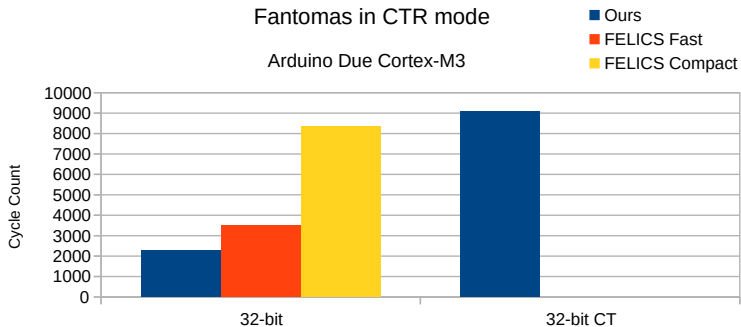
Benchmark: Encrypt+decrypt 128 bytes in CBC or encrypt 128 bits in CTR mode.

- **Related work:** FELICS (triathlon of block ciphers) [DCK⁺15].
- **Platforms:**
 1. **Cortex-M3 (Arduino Due, 32 bits):**
 - GCC 4.8.4 from Arduino with flags `-O3 -fno-schedule-insns -mcpu=cortex-m3 -mthumb`.
 - Cycles count by converting the output of the `micros()` function.

Results



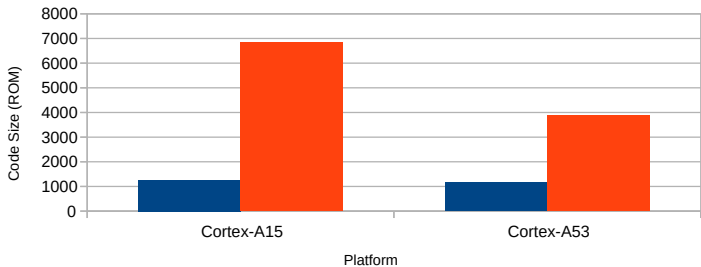
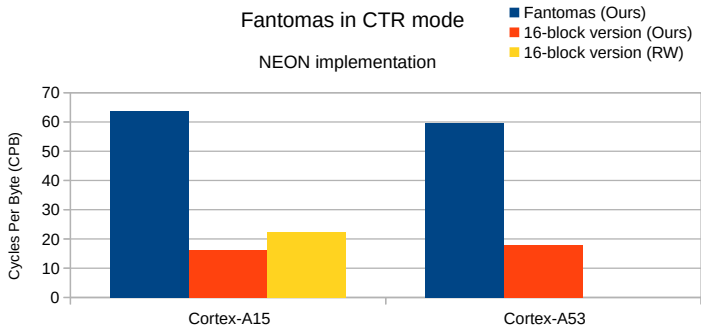
Results



Benchmark: Encrypt 128 bits in CTR mode.

- **Related work:** Adjusted timings from SCREAMv3 presentation in the CAESAR competition [GLS⁺15].
- **Platforms:**
 1. **Cortex-A15 (ODROID XU4, 32 bits + NEON):**
 - GCC 6.1.1 with flags `-O3 -fno-schedule-insns -mcpu=cortex-a15 -mthumb -march=native`.
 - Cycles count through CCNT register.
 2. **Cortex-A53 (ODROID OC2, 64 bits + NEON):**
 - GCC 6.1.1 with flags `-O3 -fno-schedule-insns -mcpu=cortex-a53 -mthumb -march=native`.
 - Cycles counts through CCNT register.

Results



Side-channel resistance

1. **Constant time implementation** against cache-timing attacks:
 - Performance penalty of **3 times** in low-end.
 - **Inherent** in vector implementations.
 - Not sufficient against **other** side-channel attacks.
2. **Masked implementation** against power attacks:
 - **Significant** quadratic performance penalty (almost twice slower with a single mask).
 - Not sufficient against **cache timing attacks**.
 - **Key masking** to force attacker to recover all shares (additional 10-20% overhead).

Fantomas has some limitations regarding side-channel resistance:

- S-boxes do not require tables, but are **expensive to mask**.
- L-boxes are cheap to mask, but **expensive to compute in constant time**.

New state-of-the-art implementations of Fantomas:

- Portable implementation in C is 35% and 52% **faster** than [DCK⁺15] on **Cortex-M**, and **similar** in code size.
- NEON implementation is 40% **faster** in ARM.

PRESENT

Proposed in 2007 and standardized by ISO/IEC, one of the first lightweight block cipher designs.

PRESENT

- Substitution-permutation network.
- **80-bit or 128-bit** key and 64-bit block.
- Key schedule for **32 rounds** with 64-bit subkeys $subkey_i$.
- 4-bit S-boxes with Boolean representation friendly to **bitslicing**.
- Bit permutation P such that $P^2 = P^{-1}$.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

Figure 1: 4-bit S-Boxes in PRESENT.

$$P(i) = \begin{cases} 16i \bmod 63 & \text{if } i \neq 63 \\ 63 & \text{if } i = 63 \end{cases}$$

PRESENT optimizations

1. Decompose permutation P^2 in **software-friendly** involutive permutations P_0 and P_1 .
2. **Rearrange** rounds to accommodate new permutations.
3. **Efficient** bitsliced S-boxes from [CHM11].
4. For CTR mode in 32 bits, process two blocks simultaneously.

Implementation

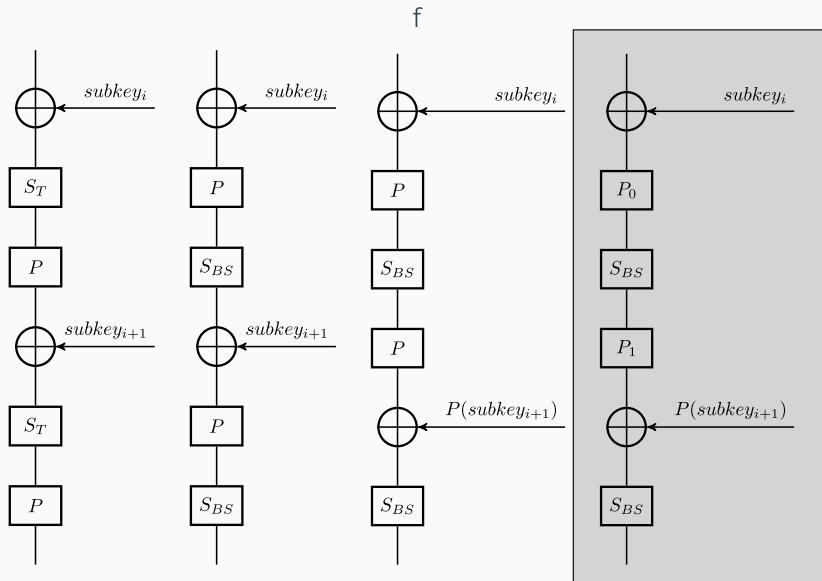
$$A = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix}$$
$$P(A) = \begin{bmatrix} 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 & 32 & 36 & 40 & 44 & 48 & 52 & 56 & 60 \\ 1 & 5 & 9 & 13 & 17 & 21 & 25 & 29 & 33 & 37 & 41 & 45 & 49 & 53 & 57 & 61 \\ 2 & 6 & 10 & 14 & 18 & 22 & 26 & 30 & 34 & 38 & 42 & 46 & 50 & 54 & 58 & 62 \\ 3 & 7 & 11 & 15 & 19 & 23 & 27 & 31 & 35 & 39 & 43 & 47 & 51 & 55 & 59 & 63 \end{bmatrix}$$

Figure 2: Permutation P in PRESENT.

$$P_0(A) = \begin{bmatrix} 0 & 16 & 32 & 48 & 4 & 20 & 36 & 52 & 8 & 24 & 40 & 56 & 12 & 28 & 44 & 60 \\ 1 & 17 & 33 & 49 & 5 & 21 & 37 & 53 & 9 & 25 & 41 & 57 & 13 & 29 & 45 & 61 \\ 2 & 18 & 34 & 50 & 6 & 22 & 38 & 54 & 10 & 26 & 42 & 58 & 14 & 30 & 46 & 62 \\ 3 & 19 & 35 & 51 & 7 & 23 & 39 & 55 & 11 & 27 & 43 & 59 & 15 & 31 & 47 & 63 \end{bmatrix}$$
$$P_1(A) = \begin{bmatrix} 0 & 1 & 2 & 3 & 16 & 17 & 18 & 19 & 32 & 33 & 34 & 35 & 48 & 49 & 50 & 51 \\ 4 & 5 & 6 & 7 & 20 & 21 & 22 & 23 & 36 & 37 & 38 & 39 & 52 & 53 & 54 & 55 \\ 8 & 9 & 10 & 11 & 24 & 25 & 26 & 27 & 40 & 41 & 42 & 43 & 56 & 57 & 58 & 59 \\ 12 & 13 & 14 & 15 & 28 & 29 & 30 & 31 & 44 & 45 & 46 & 47 & 60 & 61 & 62 & 63 \end{bmatrix}$$

Figure 3: Permutations P_0 and P_1 for optimized PRESENT.

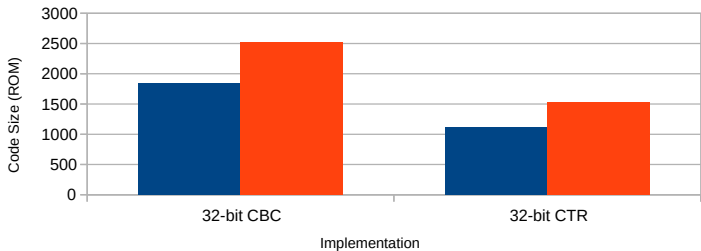
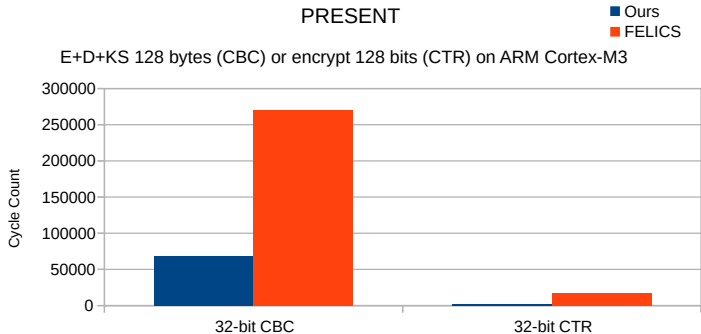
Implementation



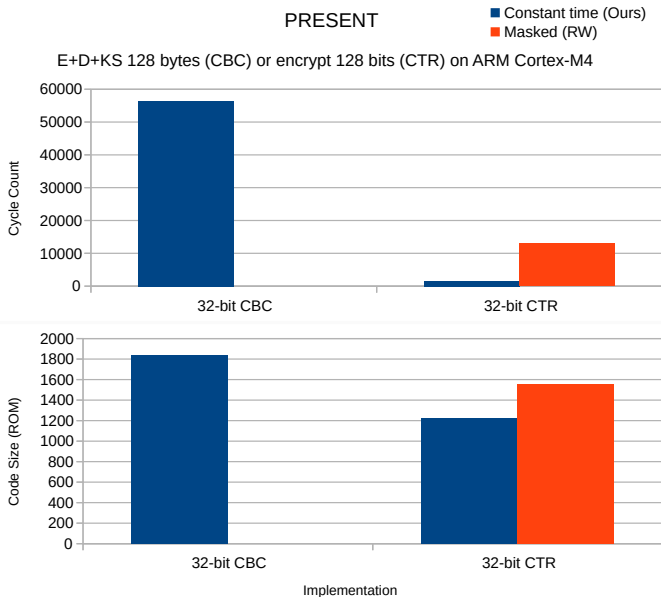
Benchmark: Encrypt+decrypt+key schedule 128 bytes in CBC or encrypt 128 bits in CTR mode.

- **Related work:** FELICS [DCK⁺15], 2nd-order constant-time masked ASM implementation of PRESENT [dGPdLP⁺16].
- **Platforms:**
 1. **Cortex-M3 (Arduino Due, 32 bits):**
 - GCC 4.8.4 from Arduino with flags `-O3 -fno-schedule-insns -mcpu=cortex-m3 -mthumb`.
 - Cycles count by converting the output of the `micros()` function.
 2. **Cortex-M4 (Teensy 3.2, 32 bits):**
 - GCC 4.8.4 from Arduino with flags `-O3 -fno-schedule-insns -mcpu=cortex-m3 -mthumb`.
 - Cycles count by converting the output of the `micros()` function.

Results



Results



Conclusions

Side-channel resistance:

- PRESENT can be efficiently implemented in **constant time**.
- Performance penalty from **masking** is lower than Fantomas, mainly due to choice of S-boxes.

New state-of-the-art implementations of PRESENT:

- S-boxes can be bitsliced (no tables) and permutations can be much faster.
- Performance improvement of **8x factor**.
- Our constant-time CTR implementation is now among the **fastest block ciphers** in the FELICS benchmark (competitive with SPARX).

Bonus Round: Efficient implementation of MAC algorithms:

- Straight-forward loop unrolling and “state unrolling” into register variables.
- Cortex-M3: improvement of Chaskey and SipHash to 7.8 and 30 cycles per byte, from 8.3 (6%) and 45 (29%) in the reference code, respectively.

Important: All timings cross-checked with the MPS2 ARM development board.

Fantomas for x86/SSE can be found at
<https://github.com/rafajunio/fantomas-x86>.

Questions?

Table 1: Comparison of block ciphers implemented in C by this work with AES in Assembly for encrypting 128 bits in CTR mode.

Block cipher	Cortex-M3		Cortex-M4		ROM
	Unprotected	CT	Unprotected	CT	
Fantomas	2291	9063	2191	7866	1272
PRESENT-80	-	2052	-	1597	1124
AES-128 [SS16]	546	1617	554	1618	12120

References I



A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe.

PRESENT: an ultra-lightweight block cipher.

In *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.



N. Courtois, D. Hulme, and T. Mourouzis.

Solving circuit optimisation problems in cryptography and cryptanalysis.

IACR Cryptology ePrint Archive, 2011:475, 2011.



D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov.

Triathlon of lightweight block ciphers for the internet of things.

IACR Cryptology ePrint Archive, 2015:209, 2015.

References II



W. de Groot, K. Papagiannopoulos, A. de La Piedra, E. Schneider, and L. Batina.

Bitsliced masking and arm: Friends or foes?

Cryptology ePrint Archive, Report 2016/946, 2016.

<http://eprint.iacr.org/2016/946>.



V. Grosso, G. Laurent, F. Standaert, K. Varici, F. Durvaux, L. Gaspar, and S. Kerckhof.

CAESAR candidate SCREAM Side-Channel Resistant Authenticated Encryption with Masking.

[http:](http://2014.diac.cr.yt.to/slides/leurent-scream.pdf)

[//2014.diac.cr.yt.to/slides/leurent-scream.pdf](http://2014.diac.cr.yt.to/slides/leurent-scream.pdf),
2015.

References III



V. Grosso, G. Leurent, F. Standaert, and K. Varici.

LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations.

In *FSE*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2014.



P. Schwabe and K. Stoffelen.

All the AES You Need on Cortex-M3 and M4.

Cryptology ePrint Archive, Report 2016/714, 2016.
<http://eprint.iacr.org/2016/714>.